Профессиональное программирование на VBA в Excel 2003

Excel 2003 Power Programming with VBA

John Walkenbach

Wiley Publishing, Inc.

Профессиональное программирование на VBA в Excel 2003

Джон Уокенбах

ДИАЛЕКТИКА

Москва · Санкт-Петербург · Киев
2005

ББК 32.973.26-018.2.75
У62
УДК 681.3.07

Компьютерное издательство "Диалектика"

Главный редактор *С.Н. Тригуб*

Зав. редакцией *В.Р. Гинзбург*

Перевод с английского и редакция *И.В. Василенко*

По общим вопросам обращайтесь в издательство "Диалектика" по адресу:
info@dialektika.com, http://www.dialektika.com

115419, Москва, а/я 783; 03150, Киев, а/я 152

Уокенбах, Джон.

У62 Профессиональное программирование на VBA в Excel 2003. : Пер. с англ. — М. : Издательский дом "Вильямс", 2005. — 800 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0771-3 (рус.)

По Excel написано немало книг. Но книга, которую вы держите в своих руках, является особенной — в ней разработка приложений электронных таблиц рассматривается в широком контексте. VBA — это всего лишь один из компонентов среды разработки пользовательских приложений, хотя и довольно существенный. Данная книга поможет вам разобраться в тонкостях разработки приложений с помощью VBA. В ней описаны многочисленные средства языка VBA, его возможности и среда использования.

Вначале вашему вниманию будет предложен обзор возможностей программы, далее вы перейдете к определению концепций VBA-программирования, а затем познакомитесь с самим языком. Если вы начинающий программист на VBA, то в данном издании найдете всю необходимую информацию, которая потребуется для дальнейшей работы. Если вы уже обладаете завидным опытом работы с VBA, то эта книга обогатит и приумножит ваши знания, пополнив их новыми методиками и примерами из реальной жизни.

ББК 32.973.26-018.2.75

ISBN 5-8459-0771-3 (рус.)
ISBN 0-7645-4072-6 (англ.)

Оглавление

Содержание

Об авторе

Джон Уокенбах — один из самых известных авторов книг по электронным таблицам. Он является главой компании JWalk and Associates Inc. — небольшой консалтинговой фирмы, расположенной в Сан-Диего и специализирующейся на разработке приложений электронных таблиц. Его перу принадлежит около 30 книг, посвященных электронным таблицам, и более 300 статей в таких журналах, как *PC World, InfoWorld, PC Magazine, Windows* и *PC/Computing*. Он также поддерживает популярный Web-узел *The Spreadsheet Page* (`www.j-walk.com/ss/`). Одно из наиболее значительных достижений Джона — разработка надстройки Power Utility Pak, получившей приз корпорации Microsoft.

Джон Уокенбах окончил университет штата Миссури, получил степень магистра и доктора философии в университете штата Монтана.

Предисловие

Добро пожаловать в книгу *Профессиональное программирование на VBA в Excel 2003*! Если вы, кроме всего прочего, разрабатываете электронные таблицы, предназначенные для других пользователей, или просто решили взять от Excel максимум возможного, то сделали правильный выбор, приобрев эту книгу.

Почему я написал эту книгу

По Excel написано немало серьезных работ. Однако данная книга является единственной, в которой разработка приложений электронных таблиц рассматривается в широком контексте. Дело в том, что VBA — всего лишь один из компонентов среды разработки пользовательских приложений (стоит заметить, что компонент этот очень существенный). А такой программный продукт, как Excel, отличается скрытыми возможностями. В нем предусмотрено немало интересных возможностей, которые находятся в глубинах, неведомых простому пользователю. Кроме того, некоторые хорошо известные средства можно использовать по-новому.

Миллионы людей по всему миру работают с Excel. Я постоянно слежу за группами новостей, посвященных электронным таблицам, и пришел к выводу, что пользователи хотят получать (и оказывать) помощь именно по тем вопросам, которым посвящена настоящая книга. Мне кажется, что лишь немногие пользователи Excel действительно понимают, каковы истинные возможности этого программного продукта. В данном издании собрана вся информация, которая поможет вам войти в эту элитную компанию. Вы готовы?

Что необходимо знать

Книга не предназначена для начинающих пользователей Excel. Если у вас нет опыта работы с этим приложением, то прочтите сначала книгу *Excel 2003. Библия пользователя* (вашего покорного слуги), которая подробно рассказывает обо всех возможностях Excel (она адресована пользователям всех уровней).

Чтобы получить от настоящей книги максимум возможного, необходимо быть достаточно опытным пользователем Excel. Я не уделяю особого внимания выполнению простых операций. Предполагается, что вы умеете следующее:

- создавать рабочие книги, вставлять листы, сохранять файлы и т.д.;
- перемещаться по рабочей книге;
- пользоваться меню и комбинациями клавиш;
- управлять панелями инструментов Excel;
- использовать функции рабочих листов Excel;
- давать имена ячейкам и диапазонам;
- применять основные возможности Windows, такие, например, как буфер обмена и приемы управления файлами.

Если вы не знаете, как все это делать, то, возможно, некоторая информация окажется для вас довольно сложной. Итак, считайте, что вас предупредили. Если же вы опытный пользователь электронных таблиц, но еще не работали с Excel 2003, то краткий обзор возможностей этого продукта можно найти в главе 2.

Что рекомендуется иметь

Для эффективной работы с книгой необходимо иметь копию Excel. Несмотря на то, что данное издание посвящено Excel 2003, часть изложенной информации также относится к Excel 97 и более поздним версиям программы. Если вы пользуетесь еще более ранней версией Excel, то эта книга точно не для вас. Кроме того, материал книги, в основном, применим и к Excel для Macintosh. Впрочем, испытания на совместимость с версией для Mac я не проводил и оставляю это для вас.

Достаточно иметь компьютерную систему на основе платформы Windows, но лучше приобрести компьютер с большим количеством оперативной памяти. Excel — это сложная программа, ее использование в низкопроизводительной системе или в системе с небольшим объемом памяти не принесет вам радости в работе.

Рекомендую устанавливать высокое разрешение экрана (800×600 — достаточно, 1024×768 — прекрасно, а 1600×1024 — это “выше крыши”). В крайнем случае, подойдет и стандартное разрешение 640×480.

Соглашения, используемые в этой книге

Уделите лишнюю минуту, чтобы просмотреть этот раздел, и узнайте о некоторых соглашениях, которые используются по всей книге.

Соглашения, относящиеся к клавиатуре

Для ввода данных вам нужна клавиатура. Кроме того, работать с меню и диалоговыми окнами также можно с помощью клавиатуры.

Ввод

Все, что вводится с клавиатуры, отображается полужирным шрифтом, например, “введите **=СУММ(B2:B50)** в ячейку `B51`”.

Длинное вводимое значение располагается в отдельной строке и для него используется моноширинный шрифт. Например, вы можете получить указание ввести следующую формулу.

```
=(СРЗНАЧ(A1:A8)+СРЗНАЧ(B1:B8))/25
```

Код VBA

В этой книге вы будете сталкиваться с фрагментами кода VBA, а также полными листингами процедур. В каждом листинге используется моноширинный шрифт, а каждая строка кода занимает в тексте книги отдельную строку. Чтобы код было легче читать, используются отступы, заданные с помощью символов табуляции. Конечно, задавать отступы не обязательно, но они помогают отделять друг от друга операторы, находящиеся рядом.

Если строка кода не помещается в одной строке книги, то используется стандартный для VBA метод продолжения строки: в конце строки вводится пробел, после которого располагается символ подчеркивания. Это означает, что данная строка кода продолжается следующим фрагментом. Например, приведенные ниже две строки текста представляют одну строку кода.

```
If Right(ActiveCell, 1) = "!" Then ActiveCell _
 = Left(ActiveCell, Len(ActiveCell) — 1)
```

Этот код можно ввести или так, как показано (т.е. в двух строках), или в одной, не используя символа подчеркивания.

ФУНКЦИИ, ИМЕНА ФАЙЛОВ И ИМЕНОВАННЫЕ ДИАПАЗОНЫ

Для отображения функций рабочих листов Excel используется верхний регистр моноширинного шрифта (например: “В ячейку `C20` введите формулу `СУММ`”). Имена, свойства, методы и объекты процедур VBA отображаются моноширинным шрифтом (“Выполните процедуру `GetTotals`”). В таких именах, чтобы они легче читались, нередко одновременно используется и нижний, и верхний регистры.

Соглашения, относящиеся к мыши

Если вы читаете эту книгу, то, значит, хорошо знаете, как пользоваться мышью. Вся “мышиная” терминология является достаточно стандартной: “указание”, “щелчок”, “щелчок правой кнопкой”, “перетаскивание” и т.д.

Что означают пиктограммы

Повсеместно в книге слева от абзацев можно встретить пиктограммы. Они используются для того, чтобы привлечь ваше внимание к особо важной информации.

Об иллюстрациях

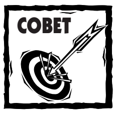
Все копии экрана, которые вы встретите в книге, получены при использовании классического интерфейса Windows XP, который мне нравится больше. Не удивляйтесь, если на экране вы увидите несколько иные окна. Содержимое диалоговых окон и файлов примеров при этом будет таким же.



Данная пиктограмма используется для того, чтобы показать, что обсуждаемый материал является новым, он появился вместе с Excel 2003. Если вы разрабатываете приложение, которое должно использоваться в более ранних версиях Excel, то обращайте на эти пиктограммы особое внимание.



Я пользуюсь этой пиктограммой, чтобы отметить важность принципа, который поможет вам легко справиться с задачей, или понять информацию, которая описывает следующий материал.



Эти пиктограммы указывают на более эффективный способ выполнить что-либо или на прием, который, возможно, не является очевидным.



Пиктограмма, используемая для описания операции, которую следует выполнять осторожно, чтобы избежать возможных проблем.



Эта пиктограмма используется для того, чтобы предоставить ссылки на другие главы, где о том или ином предмете говорится более подробно.



Этой пиктограммой обозначаются описанные в книге примеры, которые вы найдете на прилагаемом компакт-диске.

Структура книги

Главы данной книги сгруппированы в семь основных частей. Дополнительная информация приводится в нескольких приложениях.

Часть I

В этой части представлены основы изучаемой программы. В главе 1 излагается краткая история электронных таблиц — таким образом, вы сможете определить место Excel в мире программного обеспечения. В главе 2 предложен концептуальный анализ Excel 2003 — достаточно полезный для тех опытных пользователей электронных таблиц, которые только переходят к использованию Excel. Что же касается главы 3, то в ней вкратце рассматриваются формулы, кроме того, рассказывается о некоторых новых приемах. В главе 4 отмечены достоинства и недостатки разных форматов файлов, поддерживаемых и создаваемых в Excel.

Часть II

Данная часть состоит из двух глав. В главе 5 в общих чертах обсуждается тема создания приложения электронных таблиц. Глава 6 посвящена более глубокому изучению вопроса, в ней описаны типичные действия по разработке приложений электронных таблиц.

Часть III

В эту часть входят главы с 7 по 11. В ней речь пойдет о подготовке к изучению VBA. Вы ознакомитесь с VBA, с основами программирования и более подробно — с разработкой процедур и функций VBA. Глава 11 предлагает рассмотреть полезные примеры использования VBA для решения ежедневных задач.

Часть IV

В четырех главах этой части речь идет о пользовательских диалоговых окнах (называемых также пользовательскими формами `UserForm`). В главе 12 описываются альтернативные методы создания пользовательских форм. О пользовательских формах и различных элементах управления, используемых при создании этих форм, рассказывается в главе 13. В главах 14 и 15 приведены примеры пользовательских диалоговых окон, начиная от простых и заканчивая достаточно сложными.

Часть V

В этой части рассматриваются дополнительные методы программирования, которые на первый взгляд невероятно сложны для понимания. В первых трех главах говорится о том, как создавать утилиты и как использовать VBA для работы со сводными таблицами и диаграммами. В главе 19 идет речь о процессе обработки событий, который позволяет выполнять процедуры автоматически, причем выполнять именно тогда, когда произойдут определенные события. Из главы 20 вы узнаете о способах взаимодействия с другими приложениями (такими, например, как Word). Часть V заканчивается главой 21, где подробно обсуждается вопрос создания надстроек.

Часть VI

Главы этой части посвящены важным этапам создания приложений, ориентированных на конечных пользователей. В главах 22 и 23 речь идет о создании пользовательских меню и панелей инструментов. В главе 24 описаны способы подготовки интерактивной справочной системы для приложений. А в главе 25 приводится основная информация по разработке приложений, ориентированных на пользователей, а также подробно рассмотрено одно из таких приложений.

Часть VII

В пяти главах этой части освещены дополнительные вопросы, которые будут вам полезны. Информация по совместимости приведена в главе 26. В главе 27 обсуждаются разные способы применения VBA для работы с файлами. Что же касается главы 28, то в ней объясняется, как с помощью VBA управлять такими компонентами Visual Basic, как пользовательские формы и модули. В главе 29 рассмотрена тема модулей классов. Завершается данная часть полезной главой, в которой даются ответы на многие часто задаваемые вопросы о программировании в Excel.

Приложения

Завершают книгу четыре приложения. В приложении А вы найдете полезную информацию о ресурсах Internet, посвященных Excel. Приложение Б — это справочное руководство по всем ключевым словам VBA (операторам и функциям). Коды ошибок VBA приведены в приложении В. Наконец, в последнем приложении описаны файлы, которые вы найдете на прилагаемом к книге компакт-диске.

О прилагаемом компакт-диске

К данной книге прилагается компакт-диск, в котором вы найдете файлы примеров, рассматриваемых в книге. Я убежден, что любые знания лучше получать, изучая жизненные примеры, которые пригодятся при решении будущих задач. Несомненно, теоретический материал, изложенный в книге, тоже важен, но без практического применения он не стоит и ломаного гроша. На самом деле подготовка файлов примеров заняла намного больше времени, чем написание текстового материала книги.

Файлы примеров на прилагаемом компакт-диске не заархивированы, поэтому вы можете открывать их, не копируя на жесткий диск компьютера.

НА КОМПАКТ-ДИСКЕ

Детальное описание файлов на компакт-диске вы найдете в приложении Г.

Как пользоваться этой книгой

Вы можете работать с этой книгой, как вам удобно. Если захотите прочесть ее от корки до корки, то будьте моим гостем. Но так как я работаю с материалом от средней до повышенной сложности, то порядок чтения глав часто не имеет значения. Подозреваю, что большинство читателей книги будут "перескакивать" от одной части к другой, беря то тут, то там полезные "лакомые кусочки". Если же вы столкнетесь с трудной задачей, то попробуйте сперва заглянуть в оглавление — нет ли в книге решения именно вашей проблемы.

Как меня найти

Хотелось бы, чтобы вы контактировали с издателем и лично мною. После прочтения настоящей книги загляните, пожалуйста, на Web-узел издательства и оставьте свои комментарии. Давая свою оценку, будьте честными. И если вы считаете, что в определенной главе недостаточно информации — дайте мне знать. Конечно, мне бы хотелось, чтобы приходили такие отзывы, как "Это лучшая из прочитанных мною книг" или "Благодаря этой книге я получил повышение и теперь имею 105 000 долларов в год".

От тех, кто читал мои книги, я получаю каждый день по электронной почте не меньше десяти вопросов. Благодарю за сотрудничество. К сожалению, у меня просто нет времени отвечать на них. В приложении А приведен полный список источников, которые *помогут* вам в решении той или иной проблемы.

Кроме того, приглашаю вас на свой Web-узел (`http://www.j-walk.com/ss/`), где собрано довольно много материала, относящегося к Excel.

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш Web-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится вам эта книга или нет, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг. Наши координаты:

E-mail:	`info@dialektika.com`
WWW:	`http://www.dialektika.com`

Адреса для писем:

из России:	115419, Москва, а/я 783
из Украины:	03150, Киев, а/я 152

Благодарности

Во-первых, благодарю всех людей во всем мире, которые покупали предыдущие издания этой книги. Меня продолжают изумлять и вдохновлять те положительные отзывы, которые я ежедневно получаю от читателей предыдущих изданий.

Некоторые идеи, описанные в книге, принадлежат участникам ряда групп новостей, посвященных Excel. Благодарю всех, кто часто заходит на эти форумы. Ваши вопросы во многом определили практическую часть настоящей книги.

Данной книги не было бы в ваших руках, если бы не такие талантливые люди, как Сьюзен Кристоферсен (Susan Christophersen), редактор моего проекта. Особая благодарность Биллу Мэнвиллу (Bill Manville), моему техническому редактору. Билл часто сотрудничает со мной, он предложил немало прекрасных идей и неоднократно направил на правильный путь.

Джон Уокенбах
Ла-Джолла, Калифорния

Часть I

Введение в Excel

В этой части...

Глава 1

Excel 2003: исория программы

В ЭТОЙ ГЛАВЕ...

Чтобы полностью оценить те возможности разработки, которые предложены в Excel 2003, следует знать о происхождении этого программного продукта и о том, как он вписывается в общую стратегию разработки программного обеспечения компании Microsoft.

- Краткая история процессоров электронных таблиц.
- Электронные таблицы сегодня.
- Почему Excel прекрасно подходит для разработчиков.

Конечно, если последние десять лет вы работаете с персональными компьютерами и электронными таблицами, то для вас эта информация может показаться устаревшей. Но если вы собиратель пустяков, то эта глава — Клондайк. Изучив ее, вы будете "звездой" на ближайшей вечеринке компьютерных "гуру", которую удостоите своим посещением.

Краткая история процессоров электронных таблиц

Многие склонны воспринимать электронные таблицы как нечто само собой разумеющееся. На самом же деле, хотя это, возможно, и трудно осознать, были времена, когда таких таблиц не было. Тогда вместо них люди использовали громоздкие ЭВМ или калькуляторы и тратили часы на работу, которую сейчас выполняют за минуту.

Все начиналось с VisiCalc

Первый в мире процессор электронных таблиц — программа VisiCalc — был создан Дэном Бриклином (Dan Bricklin) и Бобом Фрэнкстоном (Bob Frankston) в 1978 году, когда в офисах еще даже не слыхивали о персональных компьютерах. VisiCalc была написана для компьютера Apple II — интересного маленького компьютера, игрушки по нынешним меркам. (Правда, в свое время Apple II днями напролет держал меня в состоянии гипноза.) VisiCalc, в целом, стала основой будущих электронных таблиц, а ее структуру строк и столбцов, а также синтаксис формул до сих пор можно видеть в современных электронных таблицах. VisiCalc быстро стала востребованной, и многие дальновидные компании приобретали Apple II лишь для того, чтобы создавать свои бюджетные планы с ее помощью. Со временем программе VisiCalc часто ставили в заслугу, что именно она обеспечила компьютерам Apple II бóльшую часть их первоначального успеха.

Тем временем появился новый вид персональных компьютеров; в этих ПК использовалась операционная система CP/M. Компания Sorcim разработала SuperCalc — процессор электронных таблиц, который также привлек многих последователей.

И когда в 1981 году на сцене появился компьютер IBM PC, который вывел персональные компьютеры на массовый потребительский рынок, то компания VisiCorp не стала медлить с переносом VisiCalc в эту новую аппаратную среду. Вскоре за ней последовала и Sorcim с версией SuperCalc, специально созданной для ПК.

По нынешним стандартам и VisiCalc, и SuperCalc — чрезвычайно незрелые программы. Например, текст, вводимый в ячейку, не должен был выходить за ее пределы, т.е. длинный заголовок приходилось вводить в несколько ячеек. Но как бы там ни было, возможность автоматизировать бюджетную рутину была по достоинству оценена — бумажным кассовым книгам тысячи бухгалтеров предпочли гибкие диски.

Lotus 1-2-3

Оценив успех VisiCalc, небольшая группа компьютерных гениев из компании, только что основанной в Кембридже, штат Массачусетс, решила усовершенствовать концепцию электронных таблиц. Руководимая Митчем Кэпором (Mitch Kapor) и Джонатаном Саксом (Jonathan Sachs), эта организация разработала новый продукт и провела первую в компьютерной отрасли законченную маркетинговую подготовку. Я помню, как рассматривал в *The Wall Street Journal* рекламу Lotus 1-2-3, напечатанную большим шрифтом. Это был на моей памяти первый случай, когда в издании для широкой публики появилась реклама программного продукта. Выпущенный в январе 1983 года компанией Lotus Development Corporation, этот продукт имел мгновенный успех. Несмотря на этикетку с ценой 495 долларов (да, люди действительно платили столько за программный продукт), он по продаваемости быстро обогнал VisiCalc, взлетел на вершину продаж и оставался там многие годы.

Процессор электронных таблиц Lotus 1-2-3 не только превосходил VisiCalc и SuperCalc по всем основным функциям, она также была первой программой, использовавшей новые уникальные возможности мощной 16-разрядной архитектуры IBM PC AT. Например, Lotus 1-2-3 игнорировала медленные вызовы DOS и передавала данные непосредственно в видеопамять, производя впечатление невероятной производительности системы, довольно необычной на то время. Прорывом была интерактивная справочная система, а хитроумные "движущиеся" панели с меню стали стандартом на многие годы. Впрочем, существовала главная возможность, которая действительно выделяла Lotus 1-2-3 среди остальных процессоров электронных таблиц. Речь идет о средстве создания макросов — поистине мощном инструменте, который предоставлял возможность пользователям электронных таблиц записывать осуществляемые ими операции и таким образом автоматизировать многие процессы. Когда выполнялся указанный макрос, записанные в нем операции передавались в приложение. И хотя ему не сравниться с нынешними инструментами записи, в Lotus 1-2-3 был заложен фундамент будущих технологий, интегрируемых во все процессоры электронных таблиц.

Lotus 1-2-3 — это не только первый *интегрированный* пакет, но и первый успешный среди них. В нем система производительных электронных таблиц (1) сочеталась с элементарной графикой (2) и ограниченными, но невероятно удобными средствами управления базами данных (3). Теперь понятно, что означает "легко, как 1, 2, 3"?

Компания Lotus постаралась, чтобы вслед за первым выпуском пакета Lotus 1-2-3 в апреле 1983 года последовал выпуск 1A. Этот новый программный продукт имел огромный успех и предоставил Lotus завидное положение единоличного монополиста на рынке процессоров электронных таблиц. В сентябре 1985 года выпуск 1A был заменен выпуском 2, а в июле следующего года — выпуском 2.01, содержащим исправления выявленных ошибок. Выпуск 2, в отличие от предыдущих, поддерживал

надстройки (*add-ins*) — специальные программы, которые можно интегрировать в приложение, чтобы расширить его возможности. Кроме того, в выпуске 2 содержалась усовершенствованная система управления памятью, предлагалось больше функций, максимальное количество строк увеличилось в четыре раза по сравнению с предыдущими версиями. В данной версии также поддерживался математический сопроцессор и содержался усовершенствованный макроязык, популярность которого превысила самые смелые мечты его разработчиков.

Не удивительно, что успех Lotus 1-2-3 способствовал появлению *клонов* — похожих в работе продуктов, в которых обычно предлагалось несколько дополнительных возможностей и которые, как правило, продавались намного дешевле. Среди более-менее заметных стоит упомянуть Twin компании Mosaic Software и серию VP Planner компании Paperback Software. В конце концов, за нарушение авторских прав (копирование "внешнего вида" Lotus 1-2-3) Lotus возбудила против Paperback Software судебное дело. Исход этого дела, успешный для Lotus, по существу привел к банкротству Paperback.

Летом 1989 года Lotus выпустила DOS- и OS/2-варианты долгожданной версии 3 Lotus 1-2-3. К электронным таблицам, состоящим из уже ставших привычными строк и столбцов, этот продукт добавил новое измерение; такое "расширение парадигмы" было достигнуто путем увеличения количества страниц в электронных таблицах. Впрочем, новой данная мысль в действительности не была. Идея трехмерных электронных таблиц впервые применялась в относительно малоизвестном продукте Boeing Calc, ее реализовали также в таких программах, как SuperCalc 5 и CubeCalc.

В версии 3 пакета Lotus 1-2-3 содержались многие полезные пользователям инструменты, которые, в конце концов, стали стандартными. Речь идет о многоуровневых электронных таблицах, одновременной работе с большим количеством файлов, их связывании, усовершенствованной графике и прямом доступе к внешним файлам баз данных. Однако в этой версии отсутствовала важная возможность, о которой мечтали многие пользователи: не был реализован высококачественный вывод информации.

Версия 3 начала свою жизнь с малого рыночного потенциала, поскольку требовала для нормальной работы компьютер на базе процессора 80286 с минимальным объемом оперативной памяти 1 Мбайт — требования довольно "жесткие" для 1989 года. И тут Lotus вытащила туз, припрятанный в ее корпоративном рукаве. Одновременно с объявлением о появлении версии 3 компания удивила буквально всех, заявив об усовершенствовании версии 2.01 (усовершенствованный продукт материализовался через несколько месяцев в виде Lotus 1-2-3 версии 2.2). Вопреки ожиданию большинства аналитиков, версия 3 *не* заменила версию 2. Вместо этого компания Lotus сделала блестящий ход, разбив рынок процессоров электронных таблиц на два сегмента: тот, который работает на высокопроизводительном оборудовании, и тот, для которого по карману более скромный компьютер.

Конечно, для фанатов электронных таблиц версия 2.2 продукта Lotus 1-2-3 панацеей не стала, но все-таки значительно расширила возможности пользователей. Самой важной возможностью этой версии была надстройка Allways, которая позволяла "творить" привлекательные отчеты, выполненные с использованием разнообразных шрифтов, обрамлений и заливок. Кроме того, просмотр полученных результатов на экране выполнялся в режиме WYSIWYG (What You See Is What You Get — что видишь, то и получаешь). Впрочем, когда пользователи просматривали и редактировали свою работу в этом режиме, они не могли управлять данными электронных таблиц. Но, несмотря на такое суровое ограничение, большинство пользователей Lotus 1-2-3 было вне себя от радости, потому что, имея в арсенале эту новую возможность, они наконец-то смогли создавать документы почти типографского качества.

Несколько слов по поводу защиты от копирования

На заре эры персональных компьютеров программы с защитой от копирования были правилом, а не исключением. Многие аналитики придерживаются мнения о том, что защита от копирования усложняет жизнь именно законным пользователям и мало влияет на предотвращение компьютерного пиратства.

Как вы, возможно, знаете, в Microsoft Office XP применяется технология “активации продукта”. Мишенью этой технологии являются пользователи, а предназначена она для предотвращения “случайного копирования”. Эта технология не решает более серьезную проблему — проблему борьбы с настоящими пиратами, которые создают и продают контрафактные программы.

Одной из причин, по которой компания Microsoft стала с самого начала лидировать на рынке, было отсутствие защиты от копирования в ее продуктах. Что же касается продукции ее конкурентов (Lotus 1-2-3 и WordPerfect), то такая защита была установлена. Впрочем, многие компании удостоверились, что защита от копирования не защищает производителей от посягательств пиратства, и вскоре этот процесс стал достоянием истории.

Лично мне кажется, что возврат к программам с защитой от копирования является нехорошей тенденцией. Такая защита только усложняет установку продукта и огорчает законного пользователя, если что-то идет не так. Посмотрим, насколько успешной окажется новая защита от копирования, применяемая Microsoft. Приведет ли она к увеличению продаж? Я сомневаюсь. А может, пользователи не будут менять старые версии на новые? Мне кажется, что так и будет. Заставит людей искать другой продукт? Возможно. Будет “взломана” и окажется полностью бесполезной? Вне всякого сомнения.

В мае 1990 года Microsoft выпустила Windows 3.0. Как вы, возможно, знаете, эта программа привела к изменению принципов использования персонального компьютера. Видимо, специалисты, принимавшие в Lotus решения, не считали Windows серьезным продуктом, и компания не спешила презентовать свою первую программу, работающую с электронными таблицами в Windows. Такая программа — Lotus 1-2-3 for Windows — была выпущена только в конце 1991 года. Хуже того, этот продукт, если судить объективно, оказался неудачным. Он не смог использовать преимущества среды Windows и разочаровал многих пользователей. В результате Excel, которая уже заявила о себе как о “главном” в Windows процессоре электронных таблиц, стала единоличным лидером на рынке подобных Windows-программ (и с тех пор никогда не сдавала этой позиции). Что касается Lotus, то в июне 1993 года вышла очередная ее версия: Lotus 1-2-3 версии 4 для Windows. Она была значительно лучше своего оригинала. Версия 5 этой программы для Windows появилась в середине 1994 года.

В то же время Lotus выпустила версию 4.0 этого продукта для DOS (Lotus 1-2-3 Release 4.0 for DOS). Многие аналитики (и я в том числе) ожидали появления продукта, более совместимого с Windows. Однако мы ошиблись; эта версия стала лишь более усовершенствованной по сравнению с версией 3.4. Поскольку операционная система Windows в настоящее время распространена достаточно широко, то это, скорее всего, последняя версия Lotus 1-2-3 для DOS, которая увидела свет.

Со временем электронные таблицы стали для Lotus менее важными (ее ведущим продуктом стал Notes). В середине 1995 года компания IBM приобрела Lotus Development Corporation. Появилось еще две версии Lotus 1-2-3, но это, как говорится, был тот случай, когда “и слишком мало, и слишком поздно”. Excel явно доминирует на рынке процессоров электронных таблиц, а Lotus 1-2-3 продолжает терять свои позиции.

Последние версии Lotus 1-2-3 включают такое средство, как LotusScript — язык написания сценариев, похожий на VBA. Правда, разработчики электронных таблиц не встретили это известие с особой радостью. Если можно было вернуться в прошлое, то Lotus, скорее всего, пришлось бы просто приобрести у Microsoft лицензию на VBA.

Quattro Pro

Еще одной заслуживающей внимания организацией в мире электронных таблиц является (или, следует сказать, *была*) компания Borland International. В 1994 году Novell купила у WordPerfect International и у Borland весь их бизнес, связанный с процессорами электронных таблиц. А в 1996 году и WordPerfect, и Quattro Pro были выкуплены Corel Corporation.

На ниве электронных таблиц Borland начала работать в 1987 году, выпустив продукт, называемый Quattro. Это, по сути, был клон Lotus 1-2-3, который имел несколько дополнительных средств и, возможно, более хорошую систему меню. Кроме того, указанный продукт был во много раз дешевле. Важно еще и то, что пользователи могли выбрать систему меню, похожую на применяемую в Lotus 1-2-3, и, таким образом, использовать знакомые команды, а также обеспечивать совместимость с макросами Lotus 1-2-3.

Осенью 1989 года Borland начала продавать Quattro Pro — более мощный продукт, созданный на базе, отличной от исходной Quattro, и превосходивший Lotus 1-2-3 буквально во всех областях. Например, первая Quattro Pro позволяла работать с большим количеством рабочих листов, находящихся в окнах, которые можно было перемещать и размеры которых можно было менять. Даже при том, что у него *не было* графического пользовательского интерфейса (Graphical User Interface — GUI). Еще одна деталь: Quattro Pro создавалась на основе малоизвестного продукта Surpass, приобретенного Borland.

В конце 1990 года была выпущена версия 2.0 программы Quattro Pro (Quattro Pro Version 2.0), которая уже имела поддержку трехмерной графики и обеспечивала связь с базами данных Paradox от Borland. Всего лишь полгода спустя — к большому огорчению конкурентов — появилась версия Quattro Pro 3.0, где по желанию можно было настраивать графический пользовательский интерфейс и допускалось просматривать данные в режиме слайд-шоу. Весной 1992 года появилась версия 4, в которой предлагались настраиваемые “быстрые” панели, а также новая возможность — применение аналитической графики. Что касается версии 5, вышедшей в 1994 году, она характеризовалась единственным новшеством, которое можно назвать серьезным, — наличием блокнотов рабочих листов (т.е. трехмерных рабочих листов).

Как и Lotus, компания Borland не спешила переходить на сторону Windows. Впрочем, когда осенью 1992 года Quattro Pro for Windows поступила в продажу, она составила довольно сильную конкуренцию двум другим Windows-программам, работавшим с электронными таблицами: Excel 4.0 и Lotus 1-2-3 версии 1.1 для Windows. Важно то, что в Quattro Pro для Windows предлагалась новая возможность, известная как UI Builder (построитель пользовательского интерфейса). Она позволяла разработчикам и опытным пользователям легко создавать индивидуальные варианты пользовательского интерфейса.

Кроме того, ни к чему не привела судебная тяжба между Lotus и Borland. Вначале Lotus ее выиграла, заставив Borland удалить из Quattro Pro поддержку макросов Lotus 1-2-3 и возможность создания таких же меню, как и в Lotus 1-2-3. Однако со временем, в конце 1994 года, это решение было пересмотрено, и теперь в Quattro Pro в полной мере поддерживаются средства, обеспечивающие совместимость с Lotus 1-2-3 (как будто они действительно кому-то нужны). На эту долгую борьбу обе стороны потратили миллионы долларов, а когда пыль улеглась, то настоящего победителя так и не оказалось.

Позднее Borland выпустила оригинальную версию 5 программы Quattro Pro для Windows. После того, как компания Novell получила от Borland все, что относится к процессорам электронных таблиц, версия 5 была модернизирована до версии 6. На момент написания книги текущей версией Quattro Pro является девятая, которая входит в состав WordPerfect Office 2000. Некоторые параметры этого продукта произ-

водят сильное впечатление, в том числе поддержка 1 миллиона строк и 18 278 столбцов (да, за такое большинство пользователей Excel готовы на многое). На рынке процессоров электронных таблиц Quattro Pro заслуженно занимает третье место.

Для разработчиков электронных таблиц пакет Quattro Pro долгое время был пределом совершенства. Но затем появилась Excel 5.

Microsoft Excel

А теперь перейдем к хорошему.

Многие читатели не знают, что по части электронных таблиц компания Microsoft стала приобретать опыт еще в начале 1980-х годов. И за эти годы соответствующие программы Microsoft прошли долгий путь развития: все началось с MultiPlan, отвечавшей лишь минимальным требованиям, и закончилось Excel 2003, представляющей последние разработки в этой области.

Итак, в 1982 году Microsoft выпустила программу MultiPlan — свой первый продукт для работы с электронными таблицами. Предназначенная для компьютеров, которые работают под управлением операционной системы CP/M, MultiPlan вскоре была перенесена на некоторые другие платформы, в том числе, на Apple II, Apple III, XENIX и MS DOS.

MultiPlan преимущественно игнорировала стандарты пользовательского интерфейса для программ. Трудная для изучения и применения, эта программа так никогда и не приобрела в США особой популярности. И не удивительно, что ее достаточно быстро обогнала Lotus 1-2-3.

От MultiPlan берет свое начало Excel, впервые зарекомендовавшая себя на платформе Macintosh в 1985 году. Как и все Mac-приложения, Excel являлась графической программой (в отличие от текстовой MultiPlan). В ноябре 1987 года Microsoft выпустила первую версию Excel, предназначенную для Windows (она была названа Excel 2.0 for Windows, чтобы сохранить преемственность с номером версии, выпущенной для Macintosh). Поскольку тогда операционная система Windows не имела широкого распространения, то в состав Excel 2.0 вошла исполняемая версия Windows, предназначенная исключительно для обеспечения работы Excel. Менее чем через год Microsoft выпустила новую версию Excel, версию 2.1 (Excel Version 2.1). В июле 1990 года эта компания предложила небольшое обновление (2.1b), совместимое с Windows 3.0. И хотя версии 2.*x* были по современным меркам довольно ограниченными (рис. 1.1) и не имели эффектного внешнего вида последних версий, они все равно привлекли хотя и небольшую, но верную группу поддержки и заложили прекрасный фундамент для будущих разработок. Программа Excel имела встроенный макроязык (XLM), который состоял из функций, обрабатываемых одна за другой. Этот макроязык был достаточно мощным, но трудным для изучения и применения. Как вы увидите, на смену XML пришел VBA, которому и посвящена настоящая книга.

Кроме того, Microsoft разработала версию Excel (под номером 2.20) для OS/2 Presentation Manager. Она была выпущена в сентябре 1989 года, и примерно десять месяцев спустя появилось ее обновление (версия 2.21). Впрочем, несмотря на усилия со стороны IBM, операционная система OS/2 никогда не пользовалась особой популярностью.

В декабре 1990 года Microsoft выпустила Excel 3 для Windows со значительными усовершенствованиями, как внешнего вида, так и возможностей (рис. 1.2). Среди новинок были панель инструментов, средства рисования, мощный инструмент поиска решения, поддержка надстроек, связывания и внедрения объектов (Object Linking and Embedding — OLE), трехмерные диаграммы, кнопки для макросов, упрощенная консолидация файлов, редактирование в составе рабочих групп и перенос по словам текста внутри ячейки. Кроме того, в Excel 3 существовала возможность работать с внешними базами данных (с помощью программы Q+E). Пять месяцев спустя появилось обновление Excel для OS/2.

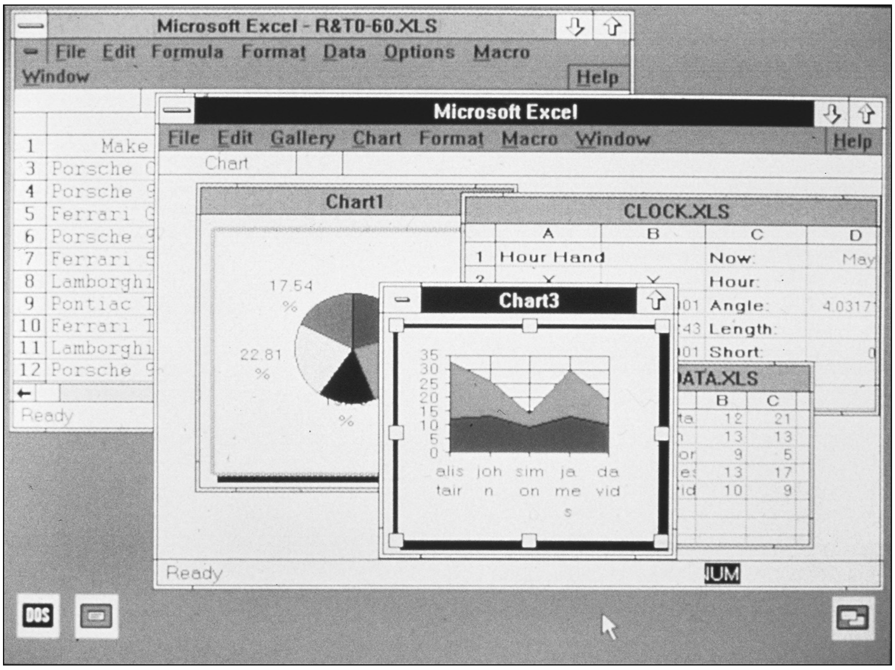


Рис. 1.1. Внешний вид Excel 2.1 для Windows. Теперь вы видите, что Excel прошла долгий путь развития? (Фотография предоставлена компанией Microsoft)

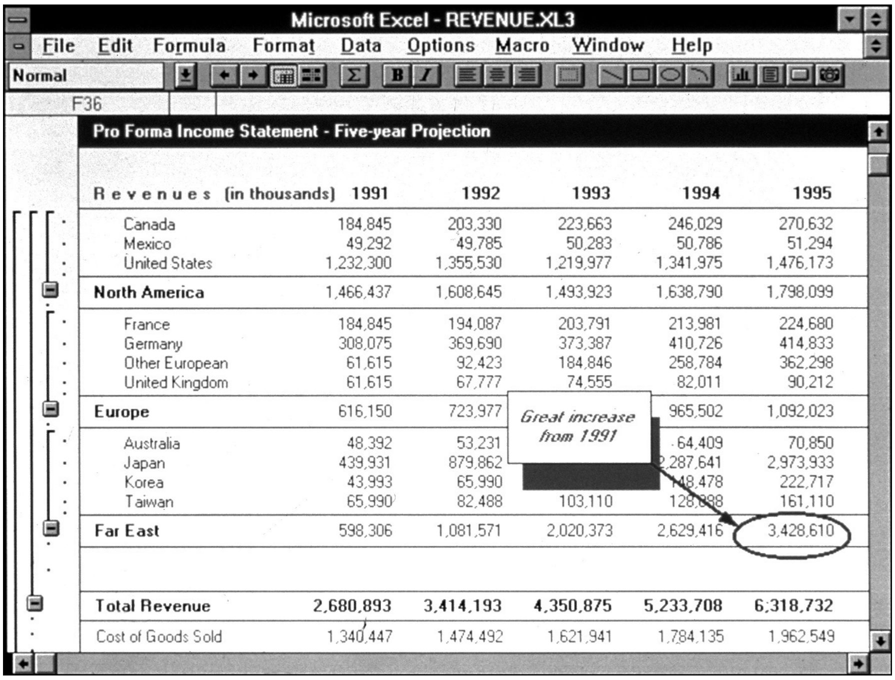


Рис. 1.3. Excel 3 была намного совершеннее первоначального выпуска. (Фотография предоставлена компанией Microsoft)

Версию 4, выпущенную весной 1992 года, было не только легче использовать, она также являлась более мощной и содержала большее количество деталей, предназначенных для опытных пользователей (рис. 1.3). Буквально в каждом обзоре компьютерных журналов, где сравнивались процессоры электронных таблиц, Excel 4 оказывалась на самом почетном месте. Тем временем отношения между Microsoft и IBM изменились к худшему; Excel 4 для операционной системы OS/2 так никогда и не была выпущена, а Microsoft прекратила выпуск версий Excel, предназначенных для этой системы.

Версия Excel 5 предстала перед публикой в начале 1994 года и сразу заслужила восторженные отзывы. Как и ее предшественница, она занимала верхнюю строчку во всех рейтингах процессоров электронных таблиц, публиковавшихся ведущими коммерческими журналами. Несмотря на жесткую конкуренцию с Lotus 1-2-3 выпуска 5 для Windows и Quattro Pro для Windows — а ведь и тот, и другой продукт мог решить буквально каждую задачу, которую подбрасывали им электронные таблицы, — Excel 5 все равно продолжала задавать тон. Кстати, эта версия была первой, в которой использовался VBA.

Версия Excel 95 (известная также как Excel 7) была выпущена одновременно с Microsoft Windows 95. Microsoft специально пропустила шестой номер, чтобы продукты, входящие в ее пакет Office, имели одинаковые номера версий. На первый взгляд, Excel 95 незначительно отличается от Excel 5. Однако существенная часть кода ее ядра была переписана, а во многих местах наблюдалось заметное увеличение быстродействия. Важно и то, что в Excel 95 использовался тот же формат файлов, что и в Excel 5. Это был первый случай, когда усовершенствованной версии Excel не представили новый формат файла. Впрочем, до конца полной совместимость не стала, поскольку в языке VBA появились некоторые усовершенствования. Следовательно, с помощью Excel 95 можно было разрабатывать приложения, которые загружались в Excel 5 (хотя и не работали там, как положено).

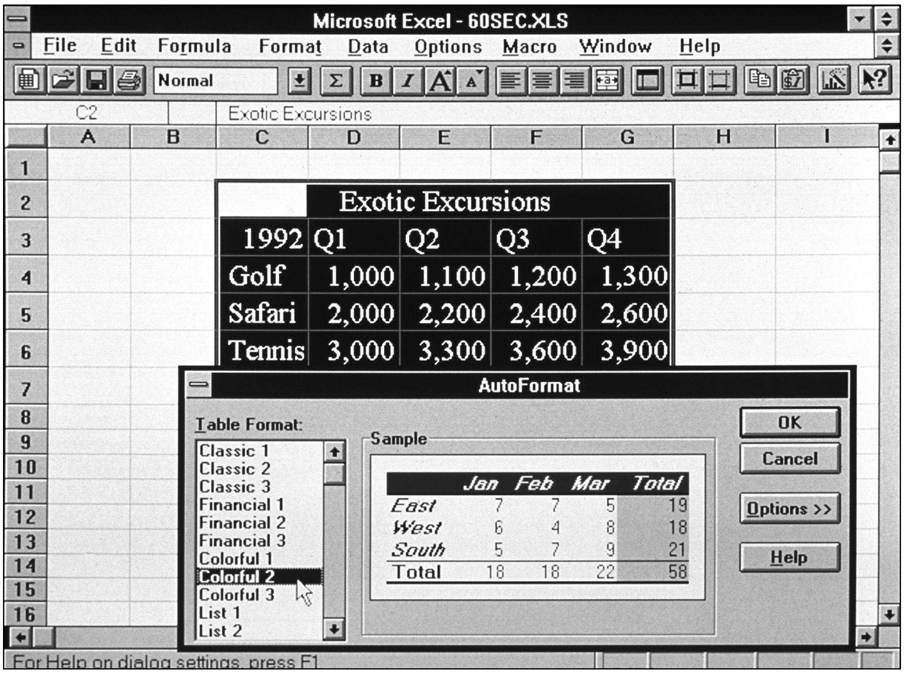


Рис. 1.3. Заметным этапом в развитии стала программа Excel 4, хотя до Excel 5 ей было далеко. (Фотография предоставлена компанией Microsoft)

В начале 1997 года Microsoft выпустила интегрированный пакет программ Office 97, в состав которого входила Excel 97. Кроме того, Excel 97 еще называется Excel 8. Эта версия характеризовалась многими общими изменениями, а также абсолютно новым интерфейсом для разработки приложений на основе VBA. Был также предложен совершенно новый способ разработки пользовательских диалоговых окон (которые теперь назывались не диалоговыми листами, а пользовательскими формами). Microsoft попыталась сделать Excel 97 совместимой с предыдущими версиями, но эта совместимость оказалась далека от совершенства. Чтобы многие приложения, разработанные с помощью Excel 5 или Excel 95, могли работать в Excel 97 или более поздних версиях, приходится прибегать к определенным уловкам.

Вопросы совместимости обсуждаются в главе 26.

Программа Excel 2000 была выпущена в начале 1999 года; она продается как часть интегрированного офисного пакета Office 2000. Усовершенствования, которые представлены в Excel 2000, относятся, в основном, к работе в Internet, хотя некоторые значительные изменения заметны и в области программирования.

Excel 2002 появилась на рынке в середине 2001 года. Как и ее предшественница, новыми возможностями, которые можно назвать серьезными, эта программа не располагает. Впрочем, появились некоторые новшества, были внесены незначительные корректировки в уже имеющиеся возможности. Вероятно, самая существенная из них — это способность восстанавливать поврежденные файлы и сохранять работу пользователя при аварийном завершении Excel. Excel продолжает доминировать на рынке и останется стандартом для пользователей любых уровней.

Текущая версия, Excel 2003, не имеет особых усовершенствований по сравнению с предыдущими версиями. Появившись в конце 2003 года, она включила всего несколько новых средств. Среди них возможность импортирования и экспортирования данных в формате XML. Кроме того, вы сможете воспользоваться тем, что в Microsoft называют "полномочиями". С их помощью определяются ограничения на использование отдельных частей рабочей книги различными пользователями. Среди незначительных улучшений стоит отметить возможность визуального сравнения двух документов на экране, улучшение функции `ПРОМЕЖУТОЧНЫЕ.ИТОГИ`, обновление справочной системы.

По определенным причинам Microsoft выпустила две подверсии Excel 2003. Поддержка "полномочий" и стандарта XML доступна только в автономной поставке Excel 2003 (а также в версии Excel 2003, включенной в состав пакета Office 2003 Professional). Поэтому, если вам необходимы указанные средства, то следите за тем, чтобы они поддерживались программой.

Почему Excel прекрасно подходит для разработчиков

В последнее время все более и более возрастет значимость разработки приложений на базе электронных таблиц. Excel 2003 — продукт с высокой степенью программируемости, поэтому, бесспорно, он является самым лучшим выбором для разработчиков таких приложений, так как поддерживает широко распространенный язык VBA.

Для разработчиков важными являются следующие возможности Excel.

- *Файловая структура.* Ориентация на многолистовую структуру позволяет легко организовывать элементы приложения и хранить его в единственном файле. Например, в единственном файле рабочей книги может находиться любое количество рабочих листов и диаграмм. Пользовательские формы и модули VBA хранятся вместе с рабочей книгой, но при этом не видны конечному пользователю.
- *Visual Basic for Application (VBA).* Этот макроязык позволяет создавать структурированные программы непосредственно в Excel. Конечно, Excel не единственный процессор электронных таблиц со структурированным языком написания сценариев (например, в Lotus 1-2-3 имеется LotusScript), но в данной программе лучше всего реализована поддержка этого языка.
- *Легкий доступ к элементам управления.* Excel позволяет довольно легко вставить в рабочий лист различные элементы управления, например, кнопки, поля со списком, переключатели и т.д. Использование таких элементов зачастую полностью освобождает от макропрограммирования.
- *Пользовательские диалоговые окна.* Вы имеете возможность создавать диалоговые окна, имеющие профессиональный внешний вид. Такая возможность Excel 2003, как пользовательские формы (впервые появившиеся в Excel 97), является намного более совершенной, чем старые диалоговые листы.
- *Пользовательские функции рабочих листов.* Для упрощения формул и вычислений вы можете с помощью VBA создавать пользовательские функции рабочих листов.
- *Настраиваемые меню.* Существует возможность вносить изменения в элементы меню, добавлять в имеющиеся меню новые элементы или создавать полностью новые меню. Другие продукты также позволяют это делать, но в Excel указанная процедура предельно упрощена.
- *Настраиваемые контекстные меню.* Excel — это единственная программа электронных таблиц, которая позволяет настраивать контекстные меню, вызываемые при щелчке правой кнопкой мыши.
- *Настраиваемые панели инструментов.* Создавать новые панели инструментов так же легко, как и другой настраиваемый элемент пользовательского интерфейса. Снова повторю, что другие продукты также позволяют это делать, но и здесь Excel оказались впереди остальных.
- *Мощные функции анализа данных.* Такая возможность Excel, как сводные таблицы, позволяет подвести итоги по довольно большому объему данных, причем особых усилий со стороны пользователей при этом не потребуется.
- *Microsoft Query.* Доступ к важным данным организовывается прямо из рабочего листа. Источниками данных могут служить базы данных стандартных форматов, текстовые файлы и Web-страницы.
- *Data Access Objects (DAO) и ActiveX Data Objects (ADO).* Эти возможности облегчают работу с внешними базами данных, выполняемую с помощью VBA.
- *Широкие возможности защиты.* Ваши приложения можно делать конфиденциальными и защищенными от изменений. Несмотря на то, что данные возможности достаточно стандартны, однако и в этом Excel имеет определенные преимущества.
- *Создание “скомпилированных” надстроек.* С помощью одной команды можно создать XLA-файлы надстроек, и эти надстройки устанавливаются без проблем.

- *Поддержка автоматизации.* Используя VBA, вы имеете возможность контролировать другие приложения, которые поддерживают автоматизацию. Например, из Excel допускается создавать отчет в Microsoft Word.
- *Возможность создания Web-страниц.* В рабочей книге Excel легко создать документ в формате HTML.

Роль Excel в стратегии Microsoft

В настоящее время большинство копий Excel продается как часть Microsoft Office — *пакета приложений*, в который входят и другие программы (какие именно, зависит от приобретаемой версии Office). Конечно, если программы могут взаимодействовать друг с другом, то это приносит пользу. Microsoft является лидером в этой области. Все продукты Office имеют удобный пользовательский интерфейс и поддерживают VBA.

Поэтому, приобретя в Excel опыт работы с VBA, вы сможете с успехом его использовать и в других приложениях — следует только изучить объектную модель этих приложений.

Глава 2

Вкратце об Excel

В ЭТОЙ ГЛАВЕ...

В этой главе представлен обзор основных компонентов Excel 2003.

- Краткое знакомство с объектной моделью Excel.
- Концептуальный обзор программы Excel; описание ее основных возможностей.
- Методы и приемы, которыми сможет воспользоваться опытный пользователь.

Глава будет особенно полезной для тех читателей, которые переходят к Excel, уже имея опыт работы в другом процессоре электронных таблиц (например, если вы опытный пользователь Lotus 1-2-3, то быстро научитесь мыслить категориями Excel). Впрочем, даже опытные пользователи Excel, просмотрев эту главу, найдут некоторые ценные советы и рекомендации.

С точки зрения объекта...

Планируя разрабатывать приложения с помощью Excel (особенно с помощью VBA), следует проанализировать понятие *объектов* — элементов Excel, которыми можно манипулировать вручную или с помощью макросов. Ниже приведены примеры объектов в Excel:

- само приложение Excel;
- рабочая книга Excel;
- рабочий лист в рабочей книге;
- диапазон ячеек в рабочем листе;
- элемент управления ListBox (Список) в пользовательской форме (в пользовательском диалоговом окне);
- лист диаграммы;
- диаграмма на листе диаграммы.

Обратите внимание, что в приведенном списке соблюдается *иерархия объектов*: объект Excel содержит объекты рабочих книг, в которых находятся объекты рабочих листов, а те, в свою очередь, включают объекты диапазонов ячеек. Подобная иерархия составляет *объектную модель* Excel. В Excel насчитывается около двухсот классов объектов, и этими объектами вы можете управлять непосредственно или с помощью VBA. Собственные объектные модели имеют и другие программные продукты Office 2003, и даже непосредственно пакет Office.

Где находятся листы модулей VBA

Язык VBA впервые появился в Excel 5. В этой версии (а также в Excel 95) модуль VBA был включен в рабочую книгу как отдельный лист. В модуле VBA, как вы, возможно, знаете, содержится VBA-код. Начиная с Excel 97, модули VBA не представлены отдельными листами. Теперь с таким модулем работают в среде VBE (Visual Basic Editor — редактор Visual Basic). Для просмотра или редактирования модуля VBA необходимо активизировать среду VBE, нажав комбинацию клавиш <Alt+F11>. Более подробно об этих модулях рассказывается в последующих главах.



Управление объектами — это фундамент разработки приложений. Из этой книги вы узнаете, как, управляя объектами Excel, автоматизировать выполнение задач, причем управлять объектами вам предстоит с помощью языка VBA. Более подробно принцип управления объектами будет рассмотрен в следующих главах.

Рабочие книги

Среди объектов Excel самым распространенным является *рабочая книга*. Все, что вы делаете в Excel, происходит в рабочей книге, которая хранится в файле, имеющем по умолчанию расширение `.xls`. В рабочей книге Excel может содержаться любое количество листов. Все они делятся на четыре вида:

- рабочие листы;
- листы диаграмм;
- листы макросов XLM (устаревшие, но до сих пор поддерживаемые);
- диалоговые листы (также устаревшие, но до сих пор поддерживаемые).

Вы можете открывать любое количество рабочих книг (каждая в своем окне), но в любой момент только одна из них может быть *активной*. Аналогично, *активным листом* может быть только один из листов рабочей книги. Чтобы активизировать лист, щелкните на его вкладке, расположенной в нижней части экрана. Для изменения имени листа дважды щелкните на вкладке и введите новое название. Если на вкладке щелкнуть правой кнопкой мыши, то появится контекстное меню.

Начиная с Excel 2002, ярлычки листов можно также выделять цветом. Для этого выполните команду Формат⇨Лист⇨Цвет ярлычка. Выделение вкладок листов цветом помогает быстро найти необходимый лист, особенно если общее их количество в книге непомерно велико.

Кроме того, окно с рабочей книгой можно скрыть. Для этого используйте команду Окно⇨Скрыть. Несмотря на то, что скрытое окно с рабочей книгой не отображается на экране, оно все равно остается открытым.

Насколько объемным может быть рабочий лист?

Всегда интересно знать ответ на этот вопрос: насколько лист может быть объемным? Выполните простой арифметический расчет (256×65 536) и тогда увидите, что в рабочем листе содержится 16 777 216 ячеек (только на одном листе). В рабочей же книге может находиться несколько рабочих листов.

Если вы используете стандартный видеорежим VGA со стандартным разрешением, то одновременно сможете увидеть 9 столбцов и 18 строк (или 162 ячейки). Это составляет менее 0,001% всего рабочего листа. Иными словами, в одном рабочем листе расположено приблизительно 104 000 экранов VGA с данными.

Если вы в каждую ячейку введете только по одной цифре (причем на введение в ячейку данных тратится одна секунда — темп достаточно быстрый), то, работая в режиме нон-стоп, сможете заполнить рабочий лист примерно за 194 дня. На распечатывание результатов ваших трудов должно пойти более 36 000 листов бумаги — пачка высотой несколько метров.

Заполнять значениями всю рабочую книгу не рекомендуется. Полученный файл будет просто огромным, к тому же работать с ним неудобно, так как системе придется постоянно сохранять информацию на диске. Как вы, возможно, догадались, программа Excel не выделяет оперативную память для каждой ячейки. Память занята данными только тех ячеек, которые действительно используются.

Рабочие листы

Самыми распространенными являются рабочие листы. Говоря об электронной таблице, пользователи обычно подразумевают рабочий лист. Рабочий лист состоит из ячеек, которые содержат данные и формулы.

В каждом рабочем листе Excel 256 столбцов и 65 536 строк. Отвечая на распространенный вопрос, скажу, что количество строк и столбцов изменить нельзя. Вы можете скрыть лишние строки и столбцы, чтобы убрать их из поля зрения, но увеличить количество строк и столбцов не в ваших силах. Возможность увеличивать количество столбцов, бесспорно, входит в пятерку самых распространенных просьб, поступающих от пользователей Excel, но Microsoft (неясно по какой причине) продолжает игнорировать эти просьбы.



В версиях, предшествовавших Excel 97, разрешалось использовать только 16 384 строки.

Предоставляемая возможность применения в рабочей книге большого количества рабочих листов ценна даже не тем, что вы получаете доступ к большему числу ячеек. Преимущество заключается в другом — большое количество рабочих листов позволит вам лучше организовать свой документ. Ранее, когда файл содержал только один рабочий лист, разработчики теряли немало времени, пытаясь организовать рабочий лист так, чтобы информация в нем хранилась наиболее рационально. Теперь вы можете хранить информацию в любом количестве рабочих листов и все равно иметь к ней мгновенный доступ (для этого потребуется щелкнуть на вкладке нужного листа).

Как вы знаете, в ячейке рабочего листа находится постоянное значение или результат выполнения формулы. В качестве значения может использоваться число, дата, булево значение ("истина" или "ложь") или текст. Кроме того, каждый рабочий лист имеет скрытый графический слой, который позволяет вставлять графические объекты (такие, например, как графики, диаграммы, чертежи, элементы управления пользовательских форм, рисунки и встроенные объекты).

Вы можете полностью контролировать ширину столбцов и высоту строк — на самом деле вы даже имеете возможность скрывать строки и столбцы (так же, как и целые рабочие листы). Текст внутри ячейки может отображаться вертикально (или под углом) и даже переноситься по словам, занимая в пределах ячейки более одной строки.

Листы диаграмм

Лист диаграммы обычно содержит одну диаграмму. Эти листы игнорируются многими пользователями, которые предпочитают сохранять диаграммы на графическом слое рабочего листа. Использовать листы диаграмм необязательно, но они облегчают печать, если на странице печатается только диаграмма. Кроме того, листы диаграмм эффективно использовать при создании презентаций.

Листы макросов XLM

Лист макросов XLM (который еще называется листом макросов MS Excel 4) в сущности является тем же рабочим листом, но со своими стандартными настройками. В частности, на листе макросов XLM отображаются сами формулы, а не их результаты. Кроме того, стандартная ширина его столбцов больше, чем у обычного рабочего листа.

Как можно понять из названия, лист макросов XLM предназначен для хранения макросов XLM. Система макросов XLM является "пережитком", доставшимся нам от предыдущих версий Excel (4.0 и более ранних). Впрочем, разработчики Excel 2003 из соображений совместимости предусмотрели поддержку макросов XLM, однако сохранить их не удается. В этой книге система макросов XLM не изучается. Основное внимание уделено более мощной системе макросов VBA.

Диалоговые листы Excel 5/95

В Excel 5 и Excel 95 пользовательское диалоговое окно создавалось путем вставки специального диалогового листа. Несмотря на то, что Excel 97 и более поздние версии также поддерживают использование этих листов, существует более удачная альтернатива — пользовательские формы (`UserForm`). В редакторе Visual Basic управление осуществляется именно пользовательскими формами.

Когда вы открываете рабочую книгу с диалоговым листом, созданным в Excel 5/95, то этот лист выглядит как лист рабочей книги.

СОВЕТ

Если из соображений совместимости вы решили вставить диалоговый лист Excel 5/95, то соответствующую команду в меню Вставка даже не ищите. Для выполнения этой операции существует только один способ — щелкнуть на вкладке любого листа правой кнопкой и выбрать в контекстном меню команду Добавить. Затем в появившемся диалоговом окне Вставка щелкните на опции Окно диалога Excel 5.0.

Учтите, в данной книге вы больше не найдете полезной информации о диалоговых листах Excel 5/95.

Пользовательский интерфейс Excel

Пользовательский интерфейс — это средство взаимодействия конечного пользователя с компьютерной программой. Пользовательский интерфейс состоит из таких элементов, как меню, панели инструментов, диалоговые окна, комбинации клавиш и т.д. В Excel для выполнения команд применяется, как правило, стандартный пользовательский интерфейс Windows, а единственное отличие состоит в том, что меню Excel — это все-таки "нестандартные" меню Windows.

Меню

Начиная с Excel 97, меню представляют собой замаскированные панели инструментов. Прямое доказательство тому — значки, которыми сопровождаются отдельные опции меню.

Система меню Excel достаточно проста. Существуют две разные строки меню (одна из них появляется, когда активен рабочий лист, а другая — когда на рабочем листе выбран объект диаграммы). В соответствии с используемыми в Windows соглашениями, недоступные команды меню будут затенены, а команды, которые открывают диалоговое окно, сопровождаются многоточием. В меню по возможности отображаются комбинации клавиш для команд (например, в меню Правка для команды Отмена указана комбинация клавиш <Ctrl+Z>).

Некоторые опции меню являются, в свою очередь, *вложенными меню*. Щелчок на такой опции приводит к отображению подменю, содержащего дополнительные команды (например, вложенным является меню, которое отображается с помощью команды Правка⇨Заполнить). Вложенные меню вы узнаете по маленькой стрелке, указывающей вправо.

Пользователь или разработчик может перенастраивать по своему усмотрению структуру меню. Для этого выполните команду Вид⇨Панели инструментов⇨Настройка. Важно понимать, что изменения в меню, сделанные таким образом, являются “постоянными”. Другими словами, даже если закрыть приложение Excel и запустить его заново, то изменения в меню останутся в силе. В этом заключается *главное* отличие такого способа от вызова редактора меню, поддерживаемого в Excel 5 и Excel 95, но отсутствующего в Excel 2002 и 2003.

Чтобы изменить меню, созданное с помощью редактора меню Excel 5 или Excel 95, используйте программы Excel 5 или Excel 95. Существует еще один способ — установить специальную утилиту, позволяющую выполнить необходимые операции.

Контекстные меню

В Excel также используются контекстные меню, которые появляются при щелчке правой кнопкой мыши на объекте или наборе выделенных объектов. Обратите внимание на то, что каждое такое меню может настраиваться разработчиком или конечным пользователем.

С помощью VBA вы имеете возможность управлять контекстными меню произвольным образом.

Более подробно о настройке меню рассказывается в главе 23.

Панели инструментов

Программа Excel 2003 насчитывает более десяти заранее подготовленных панелей инструментов (две из них используются как меню). Кроме того, вы можете создать столько новых панелей инструментов, сколько захотите. Для настройки имеющихся панелей инструментов или создания новых используйте команду Вид⇨Панели инструментов⇨Настройка. Вам предоставлена возможность распространять настроенные панели инструментов среди пользователей, вкладывая их в рабочие книги.

Панели инструментов можно *прикреплять* (размещая вдоль любого края экрана) или делать их *плавающими*. В Excel панели инструментов Стандартная и Форматирование по умолчанию закрепляются непосредственно под строкой меню.

Кнопки панелей инструментов при отображении могут иметь размеры одного из двух видов — хотя, на наш взгляд, кнопки бо́льших размеров не очень привлекательны и удобны. В Excel встроен простой, однако достаточно эффективный редактор кнопок панелей инструментов (рис. 2.1). Впрочем, в Excel для кнопок панелей инструментов подготовлен определенный набор изображений, поэтому вам, скорее всего, редактор кнопок не понадобится.

О панелях инструментов подробно рассказывается в главе 22.

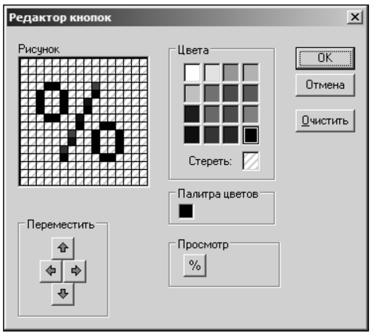


Рис. 2.1. Редактор кнопок, предназначенных для панелей инструментов, не представляет собой ничего особенного, однако с возложенными обязанностями справляется прекрасно

Диалоговые окна

При выполнении большинства команд Excel отображаются диалоговые окна. По принципу своей работы они практически одинаковы, если не считать некоторых отличий, которыми характеризуются диалоговые окна надстроек сторонних производителей.

Большинство диалоговых окон — *модальные*. Это означает, что вы должны закрыть диалоговое окно для доступа к данным рабочего листа. Тем не менее, отдельные диалоговые окна *немодальные*. К ним относится диалоговое окно поиска и замены слов, которое отображается при выборе команды Правка⇨Найти.

В некоторых диалоговых окнах Excel используется нечто похожее на вкладки записной книжки. Благодаря им одно диалоговое окно заменяет нескольких равнозначных. Первый пример диалогового окна, имеющего вкладки, — это Параметры (рис. 2.2). В данном диалоговом окне представлено 13 вкладок. Чтобы оно появилось на экране, выполните команду Сервис⇨Параметры.

Значительным усовершенствованием является применение пользовательских форм (впервые представленных в Excel 97). Эти формы необходимы разработчику для создания сложных диалоговых окон, в том числе содержащих вкладки (для этого используется элемент управления MultiPage).

ПЕРЕКРЕСТНАЯ ССЫЛКА

Более подробно о создании пользовательских форм и о принципах работы с ними рассказывается в части IV.

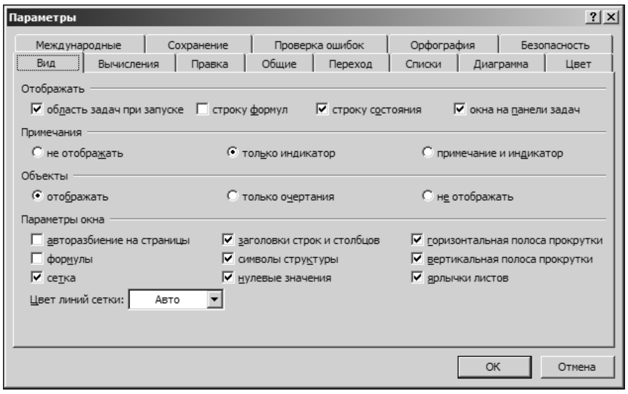


Рис. 2.2. Диалоговые окна, имеющие вкладки, обеспечивают доступ к ряду параметров, причем не отличаясь особым многообразием

Перетаскивание

Такая возможность пользовательского интерфейса Excel, как перетаскивание, позволяет свободно перемещать объекты, расположенные на графическом слое, и таким образом изменять расположение этих объектов. Если при перетаскивании удерживать кнопку <Ctrl>, то выбранные объекты будут дублироваться.

Кроме того, в Excel операции перетаскивания можно выполнять также над отдельными ячейками и их диапазонами — ячейку или диапазон легко переместить в другое место. А если при перетаскивании удерживать нажатой кнопку <Ctrl>, то выбранный диапазон будет копироваться.



Возможность перетаскивания не является обязательной; ее можно отключить на вкладке Правка диалогового окна Параметры.

Существует возможность перетащить диапазон и на рабочий стол Windows, создав таким образом файл фрагмента. Впоследствии этот фрагмент можно перетащить в другую рабочую книгу (или другое приложение) и вставить как OLE-объект.

Комбинации клавиш

В приложении Excel существует достаточно много комбинаций клавиш. Например, чтобы копировать выделение, нажмите <Ctrl+C>. Если вы начинающий пользователь Excel или хотите повысить скорость выполнения операций в программе, то рекомендуем просмотреть разделы справочной системы, посвященные комбинациям клавиш. Изучение комбинаций клавиш — ключ к успешному использованию возможностей Excel. В файлах справочной системы приведены таблицы, в которых собраны все полезные советы по использованию клавиатуры для выполнения часто выполняемых операций.

Смарт-теги

Смарт-тег — это небольшой значок, который автоматически появляется на рабочем листе при выполнении определенных действий. После щелчка на сматр-теге на экране отображается своеобразное контекстное меню, в котором перечислены специальные команды. Например, при копировании и вставке данных на рабочий лист вы увидите смарт-тег в правом нижнем углу вставляемого диапазона (рис. 2.3).

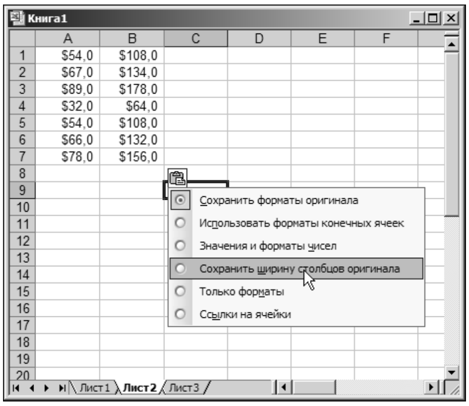


Рис. 2.3. Смарт-тег появляется также при копировании и вставке данных

Вы найдете в Excel очень много самых разных смарт-тегов. После установки надстроек независимых производителей их число может увеличиться. Не всем нравятся смарт-теги, поэтому в Excel их можно отключить. Для этого выберите команду Сервис⇨Параметры автозамены и перейдите на вкладку Смарт-теги.

Область задач

Впервые область задач появилась в Excel 2002. Эта многофункциональное средство, прикрепляемое к правому краю окна, значительно улучшило пользовательский интерфейс программы. Область задач используется для выполнения самых разных операций. Среди них отображение справочных сведений, поиск информации, управление буфером обмена, импорт и экспорт XML-данных и т.д.

НОВИНКА

В Excel 2003 функциональные возможности области задач значительно расширены.

Ввод данных

Вводить данные в среде Excel достаточно просто. Каждое введенное в ячейку значение интерпретируется программой Excel как элемент списка:

- числовое значение (им может быть значение даты и/или времени);
- текст;
- формула;
- булево значение (“истина” или “ложь”).

Советы по введению данных

Следующие советы по введению данных особенно пригодятся тем, кто переходит к использованию Excel, имея опыт работы в других процессорах электронных таблиц.

- Если перед вводом данных вами был выбран диапазон ячеек, то для завершения ввода значения в одну ячейку и перехода к следующей нажмите клавишу <Enter>. Аналогично, для перемещения вверх используйте комбинацию клавиш <Shift+Enter>, для перемещения вправо — клавишу <Tab>, а для перемещения влево — комбинацию клавиш <Shift+Tab>.
- Чтобы после ввода данных не нажимать клавиши со стрелками для перехода к следующей ячейке, на вкладке Правка диалогового окна Параметры установите флажок Переход к другой ячейке после ввода. Доступ к этому диалоговому окну можно получить, выполнив команду Сервис⇨Параметры. Кроме того, вы имеете возможность указать направление, в котором выполняется переход к следующей ячейке.
- Если в каждую ячейку диапазона требуется ввести одни и те же данные, то выделите необходимый диапазон, введите информацию в активную ячейку, а затем нажмите комбинацию клавиш <Ctrl+Enter>.
- Чтобы скопировать содержимое активной ячейки во все остальные ячейки выбранного диапазона, нажмите клавишу <F2>, а затем — комбинацию клавиш <Ctrl+Enter>.
- Если нужно заполнить диапазон значениями, возрастающими в каждой следующей ячейке на постоянный инкремент, то, нажав клавишу <Ctrl>, перетащите маркер заполнения в нижний правый угол выделения.
- Чтобы создать пользовательский список автозаполнения, перейдите на вкладку Списки диалогового окна Параметры.

- Если ячейку необходимо скопировать без увеличения значения на постоянный инкремент, то перетащите маркер заполнения в соответствующий угол выделения. Кроме того, можете нажать <Ctrl+D>, чтобы скопировать ячейку вниз, или <Ctrl+R> — чтобы скопировать ее вправо.
- Внутри ячейки допускается использование символов табуляции и возврата каретки (чтобы расположенный в ней текст легче было воспринимать). Ввести символ табуляции можно, нажав <Ctrl+Alt+Tab>. А чтобы ввести символ возврата каретки, нажмите <Alt+Enter>. Символы возврата каретки предоставляют возможность разбить содержимое ячейки на строки внутри одной ячейки.
- Для ввода дроби нажмите 0, затем — клавишу пробела, после чего введите саму дробь (используя клавишу </>). Тогда содержимое ячейки приобретет дробный числовой формат.
- Чтобы автоматически задать для ячейки денежный формат, перед хранящимся в ней значением введите символ денежной единицы (в США это знак доллара). Для введения значения в процентном формате после значения введите знак процента; для разделения разрядов можете использовать соответствующие символы, задаваемые национальными стандартами.
- Нажмите <Ctrl+;> для ввода в ячейку текущей даты или <Ctrl+Shift+;> для ввода в ячейку текущего времени.
- Чтобы ячейка или диапазон принимали значения только определенного типа (или значения в пределах определенного диапазона), используйте команду Данные⇨Проверка.

Формулы всегда начинаются со знака равенства (=). Впрочем, программа Excel также приспособлена для пользователей, привыкших к Lotus 1-2-3, и в качестве первого символа формулы адекватно принимает амперсанд (&), знак “плюс” (+) или “минус” (−). После того, как вы нажмете <Enter>, введенный вами первый символ автоматически будет заменен на знак равенства.

Формулы, имена и функции

Формулы — это те элементы, благодаря которым электронная таблица оправдывает свое название. В Excel с формулами связаны невероятные возможности, о которых следует знать каждому. Вы также можете создавать формулы массивов, использовать оператор пересечения, вставлять в них ссылки и создавать мегаформулы (этим термином я обозначаю длинные и невразумительные, зато очень эффективные формулы).

ПЕРЕКРЕСТНАЯ ССЫЛКА

О формулах рассказывается в главе 3, в которой приведен ряд советов и рекомендаций.

Кроме того, в Excel существуют и другие средства, которые помогают найти ошибки или проследить логику незнакомой электронной таблицы. Чтобы получить доступ к этим инструментам, выполните команду Сервис⇨Зависимости формул.

В Excel 2002 инструменты проверки формул приобрели тот совершенный вид, с которым вы будете сталкиваться. Так, например, теперь программа автоматически ищет потенциально неправильные формулы, о чем незамедлительно извещает пользователя (рис. 2.4).

Функции рабочих листов позволяют проводить такие вычисления или операции, которые выполнить по-другому просто невозможно. Программа Excel располагает большим количеством встроенных функций. Более того, установив надстройку Analysis ToolPack, вы получите в свое распоряжение еще и другие функции (многие из которых довольно экзотические).

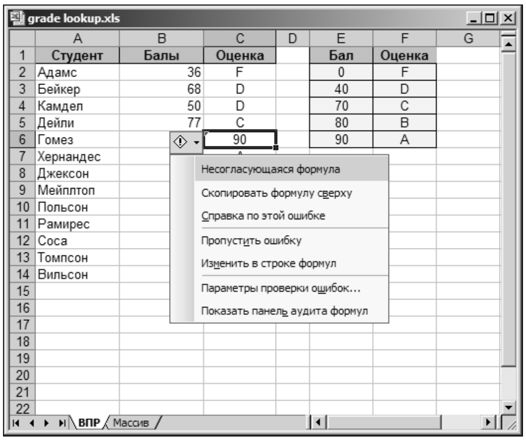


Рис. 2.4. Проверка формул на наличие ошибок

Самый легкий способ найти необходимую функцию — использовать диалоговое окно Мастер функций, которое показано на рис. 2.5. Это диалоговое окно появляется по щелчку на кнопке Вставка функции, которая находится в строке формул (вы также можете выполнить команду Вставка⇨Функция или же нажать комбинацию клавиш <Shift+F3>). Если вам данная возможность программы неизвестна, то советуем в ближайшее время с ней ознакомиться — вы узнаете много нового и ценного.

В Excel 2002 впервые появилась возможность идентификации по ключевому слову, используемая в случае, если вы забыли имя функции. Например, вам необходимо найти функцию, которая преобразует текст в код ASCII — проведите поиск по коду, а затем щелкните на кнопке Найти.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Кроме того, в Excel существует возможность создавать на языке VBA собственные функции рабочих листов (см. главу 10).

Имя — это идентификатор, который позволяет ссылаться на ячейку, диапазон, значение, формулу или графический объект. Формулы, в которых используются имена, воспринимать намного легче, чем написанные с помощью ссылок на ячейки. Более того, формулы с именованными ссылками создавать намного проще.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Об именах речь пойдет в главе 3.

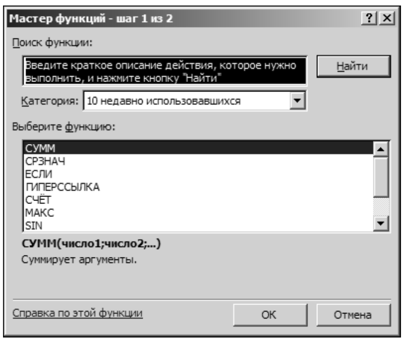


Рис. 2.5. Для вставки функции в формулу лучше всего использовать диалоговое окно Мастер функции

Настройка вида

Если говорить о настройке отображаемого на экране (строка состояния, строка формул, панели инструментов и т.п.) вида, то можно утверждать, что вы имеете довольно большой выбор. Например, выполняя команду Вид⇨Во весь экран, вы избавитесь от дополнительных графических инструментов, за исключением строки меню, и таким образом максимально расширите рабочую область окна программы. Кроме того, с помощью вкладки Вид диалогового окна Параметры, вы можете настроить все отображаемые в окне рабочего листа объекты (и даже скрыть полосы прокрутки и линии сетки).

Excel позволяет разрабатывать также приложения, которые могут и не выглядеть как электронная таблица.

Выделение объектов

Обычно выделение объектов выполняется с помощью стандартных методов, принятых в Windows. Диапазон ячеек можно выделить с помощью мыши, щелкнув и затем обведя необходимые ячейки. Если щелкнуть на объекте, который расположен на графическом слое, то объект будет выделен. Чтобы выделить ряд объектов или несмежных ячеек, при выделении каждого из них нажмите клавишу <Ctrl>. Если следует выделить большой диапазон, щелкните на ячейке, расположенной в одном из углов этого диапазона, прокрутите документ до противоположного угла диапазона, а затем, нажав <Shift>, щелкните мышью на последней ячейке диапазона.

НА ЗАМЕТКУ

В более ранних, чем Excel 97, версиях после щелчка на внедренной диаграмме происходило выделение всей диаграммы. Начиная же с Excel 97, щелчок на диаграмме приводит к выделению одного из ее объектов. Поэтому чтобы выделить объект всей диаграммы, при щелчке на ней удерживайте клавишу <Ctrl>.

Форматирование

В Excel выполняется форматирование двух видов: числовое и стилистическое.

Числовое форматирование

Числовым форматом называют тот "вид", который приобретает значение в ячейке. Кроме выбора формата из заранее определенного списка, вы можете создать собственный (рис. 2.6). Эта процедура подробно объяснена в справочной системе программы.

Некоторые числовые форматы задаются автоматически, в зависимости от вводимого значения. Например, если введено значение с принятым у вас символом денежной единицы (в США таким символом является знак доллара), то будет использован числовой денежный формат.

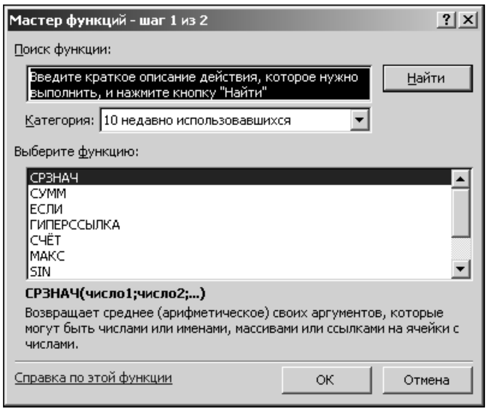


Рис. 2.6. В Excel существует достаточно большой выбор числовых форматов

Стилистическое форматирование

Стилистическим называется форматирование, применяемое с целью улучшения внешнего вида вашей работы. Многие кнопки на панели инструментов обеспечивают прямой доступ к основным возможностям форматирования, независимо от того, работаете вы с ячейками, нарисованными объектами или диаграммами. Например, с помощью кнопки Цвет заливки вы можете поменять цвет фона ячейки и заливку в нарисованном текстовом блоке, а также цвет любой полосы диаграммы. Но полноценное форматирование вы имеете возможность применить только в диалоговом окне Формат ячеек.

Самый легкий способ вывести необходимое диалоговое окно и задать формат объекта заключается в следующем: выделите сам объект и нажмите комбинацию клавиш <Ctrl+1>. Вы также можете щелкнуть на объекте правой кнопкой мыши и выбрать из контекстного меню команду Формат *xxx* (где символы *xxx* — это название выделенного объекта). В результате появится диалоговое окно, имеющее несколько вкладок. В этом окне можно задать любое форматирование, которое только возможно назначить для выделенного объекта.

В Excel часто используется такая возможность, как условное форматирование. Доступ к ней вы получите, если выполните команду Формат⇨Условное форматирование. Эта возможность позволяет задать форматирование, которое будет применяться только при определенных условиях. Например, можно выделить другим цветом те ячейки, значения в которых превышают указанную величину.

Фигуры

Как уже отмечалось ранее, в каждом рабочем листе находится скрытый графический слой, на котором могут располагаться диаграммы, изображения, элементы управления (такие, например, как кнопки и списки), а также фигуры.

В Excel, благодаря кнопкам на панели инструментов Рисование, можно непосредственно на рабочем листе нарисовать большое количество различных геометрических фигур. Помните также, что группу объектов вы вправе сгруппировать в один общий объект, для которого несложно поменять размеры и местоположение.

О некоторых нарисованных объектах следует рассказать более подробно.

- С помощью панели инструментов Рисование можно вставлять так называемые автофигуры (рис. 2.7), которые представлены достаточно широко. Как только выбранная фигура окажется на рабочем листе, ее можно видоизменить: выделите ее и перетащите маркеры ограничивающей рамки. Кроме того, добавьте к фигуре падающую тень, текст или трехмерные эффекты.
- *Текстовый блок* позволяет отображать текст, который не привязывается к границам строк и столбцов, а это хороший способ вставлять подписи к строкам и столбцам.
- По непонятной причине разработчики Excel сделали так, что создать *связанный объект изображения* не очень просто. Скопируйте диапазон и выполните команду Правка⇨Вставить связь с рисунком (появляется в меню Правка только при нажатии клавиши <Shift>). Команда Вставить связь с рисунком используется тогда, когда необходимо распечатать выделенные несмежные диапазоны. Например, вы можете сделать “снимок” выбранных диапазонов, затем вставить полученные рисунки в отдельную область и распечатать ее.
- Многие элементы управления, применяемые в пользовательских диалоговых окнах, можно размещать непосредственно на рабочем листе, что позволит сделать их намного более удобными, сведя к минимуму потребность в пользовательских диалоговых окнах.

Достаточно новый и непривычный тип фигур в Excel — это организационные диаграммы. Чтобы выбрать один из шести типов организационных диаграмм, выполните команду Вставка⇨Схематическая диаграмма (рис. 2.7). После того, как диаграмма будет вставлена, в ней можно проводить простые изменения, используя для этого панель инструментов Диаграммы.



Рис. 2.7. Автофигуры и организационные диаграммы — это самые обычные объекта рабочих листов

Диаграммы

Программа Excel предоставляет возможность управления диаграммами. Как уже отмечалось, диаграммы можно или размещать на специальном листе, или прикреплять к рабочему листу с данными.

Самый легкий способ создания диаграммы на основе имеющихся данных — выделить эти данные, а затем использовать мастер создания диаграмм (для этого на панели инструментов Стандартная щелкните на кнопке Мастер диаграмм). Этот мастер в пошаговом режиме “проведет” вас по всем этапам создания диаграммы (рис. 2.8).

Вы также имеете возможность создавать диаграммы сводных таблиц. Такая диаграмма связана с соответствующей сводной таблицей и позволяет просматривать в графическом виде сводные данные — при этом используются те же методы управления, что и в самой сводной таблице.

Настройка диаграммы — это еще один “конек” Excel. Для выделения незакрепленной диаграммы просто щелкните на ней. Двойной щелчок приводит к отображению диалогового окна настройки диаграммы или соответствующего элемента.

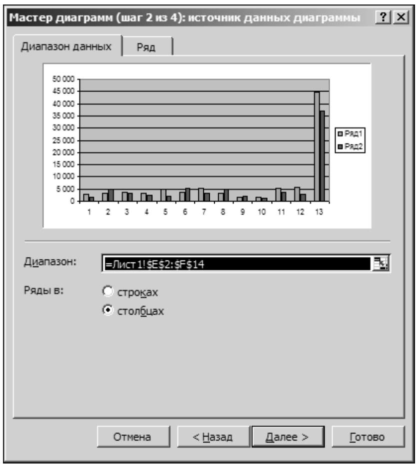


Рис. 2.8. Для создания диаграмм используется специальный мастер

Макросы и программирование

Во всех популярных процессорах электронных таблиц поддерживается собственный макроязык. В Excel их два: XLM и VBA. Первоначально используемый макроязык XLM давно устарел. Ныне он заменен на VBA. Следует отметить, что в Excel будет выполняться любой макрос XLM, однако записать такие макросы не представляется возможным. Поэтому для создания новых макросов придется использовать исключительно VBA.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Языку VBA посвящена часть III.

Доступ к базам данных

В течение многих лет процессоры электронных таблиц предоставляли пользователям возможность работать с простыми таблицами так называемых *плоских баз данных* (эта возможность даже поддерживалась в первой версии Lotus 1-2-3). Инструменты управления плоскими базами данных существует также в Excel.

Базы данных, создаваемые в процессорах электронных таблиц, делятся на две категории.

- *Базы данных рабочих листов.* Вся база данных хранится на рабочем листе, ограничивающем ее размеры. В Excel она не может иметь более 65 535 записей и 256 полей (в самой верхней строке указываются имена полей).
- *Внешние базы данных.* Данные хранятся в одном или нескольких файлах на диске и загружаются по мере необходимости.

Базы данных рабочих листов

Обычно, когда указатель находится внутри базы данных, Excel распознает ее и по мере возможности отображает имена полей. Например, если вы переместите указатель внутрь базы данных, которая находится на рабочем листе, и затем выполните команду Данные⇨Сортировка, то сможете указать критерии сортировки, выбрав в раскрывающихся списках необходимые имена полей.

Особенно полезной является такая возможность Excel, как автофильтр — он позволяет отображать только те записи, которые вы хотите видеть на экране. Когда включен этот режим, вы можете фильтровать данные, выбирая значения из раскрывающихся списков. Данные списки появляются там, где введены имена полей. Для активизации автофильтра следует выполнить команду Данные⇨Фильтр⇨Автофильтр. Строки, не удовлетворяющие условию фильтрации, будут временно скрыты. На рис. 2.9 показан один из примеров того, как используется автофильтр в реальном рабочем листе.

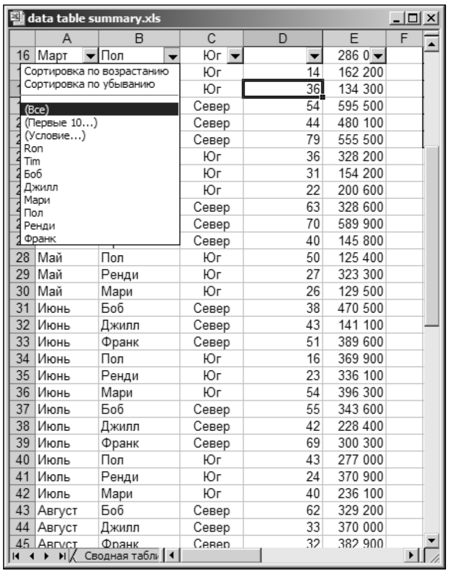


Рис. 2.9. В Excel с помощью автофильтра можно просматривать только те записи базы данных, которые удовлетворяют вашим критериям

Если при управлении базами данных электронных таблиц вы предпочитаете использовать те традиционные приемы, в которых задействуется несколько критериев, выполните команду Данные⇨Фильтр⇨Расширенный фильтр.

НОВИНКА

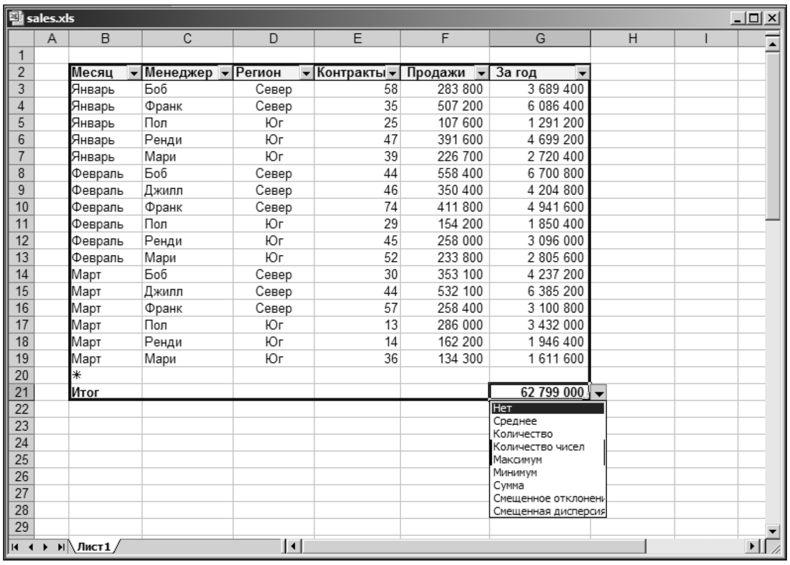
В Excel 2003 появилась специальная команда преобразования выделенного диапазона в список. Для ее использования выберите Данные⇨Список⇨Создать список. После создания списка автофильтр включается автоматически, а в конце списка отображается строка итогов (рис. 2.10). Все, что вам необходимо, — это выбрать итоговую функцию. Стоит заметить, многие пользователи не считают подобное средство улучшением, поскольку его возможности сильно ограничены.

Внешние базы данных

Чтобы иметь возможность работать с таблицами внешних баз данных, выполните команду Данные⇨Импорт внешних данных⇨Создать запрос. После запуска программы Microsoft Query вы сможете выбрать исходные базы данных, а также определить запросы к ним. Результаты выполнения запросов передаются непосредственно на рабочий лист.

Используя технологии DAO (Data Access Objects — объекты доступа к данным) и ADO (ActiveX Data Objects — объекты данных ActiveX), вы имеете возможность работать с объектами данных, созданных вне Excel. Обе эти технологии позволяют с помощью VBA получить доступ к внешним базам данных, и, кроме того, ADO используется для изменения данных непосредственно в базе данных, а не только на рабочем листе.

Вы также можете создавать Web-запросы, чтобы получать данные, хранящиеся в корпоративной сети или в глобальной сети Internet.



Месяц	Менеджер	Регион	Контракты	Продажи	За год
Январь	Боб	Север	58	283 800	3 689 400
Январь	Франк	Север	35	507 200	6 086 400
Январь	Пол	Юг	25	107 600	1 291 200
Январь	Ренди	Юг	47	391 600	4 699 200
Январь	Мари	Юг	39	226 700	2 720 400
Февраль	Боб	Север	44	558 400	6 700 800
Февраль	Джилл	Север	46	350 400	4 204 800
Февраль	Франк	Север	74	411 800	4 941 600
Февраль	Пол	Юг	29	154 200	1 850 400
Февраль	Ренди	Юг	45	258 000	3 096 000
Февраль	Мари	Юг	52	233 800	2 805 600
Март	Боб	Север	30	353 100	4 237 200
Март	Джилл	Север	44	532 100	6 385 200
Март	Франк	Север	57	258 400	3 100 800
Март	Пол	Юг	13	286 000	3 432 000
Март	Ренди	Юг	14	162 200	1 946 400
Март	Мари	Юг	36	134 300	1 611 600
*					
Итог					62 799 000

Рис. 2.10. Автоматически созданный список позволяет подводить итоги с помощью всего нескольких простых функций

Функции использования Internet

В Excel представлен набор функций, помогающих управлять ресурсами Internet, например, возможность сохранить рабочий лист или всю рабочую книгу в формате HTML, поддерживаемом Web-браузерами. Кроме того, непосредственно в ячейки можно вставлять гиперссылки, активизируемые щелчком мыши (в том числе и адреса электронной почты).

Файлы Excel рекомендуется сохранять в формате HTML, что обеспечивает значительную интерактивность. Эта возможность, представленная благодаря Web-компонентам пакета Office, позволяет публиковать на Web-сервере интерактивные рабочие книги и дает возможность работать с ними другим пользователям (которые имеют лицензию на Web-компоненты пакета Office).

Детально о сохранении данных в формате HTML мы поговорим в главе 4.

Поддержка XML

Поддержка этого стандарта впервые реализована в Excel 2003. С помощью специальных команд программы вы сможете импортировать XML-данные и поместить их в ячейки рабочего листа.

Управлять XML-данными в Excel 2003 вы сможете только в профессиональном выпуске программы.

В главе 4 вы найдете детальное описание средств XML.

Инструменты анализа

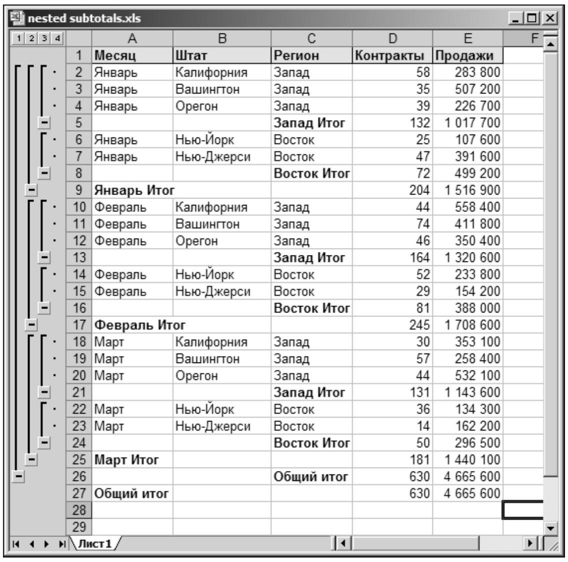
Что касается анализа, то Excel справляется с ним довольно неплохо (именно по этой причине большинство пользователей обращаются к электронным таблицам). Некоторые задачи по анализу данных решаются с помощью формул, однако Excel предлагает и другие варианты.

Структуры

Режим структуры рабочей таблицы зачастую является эффективным средством управления иерархически упорядоченными данными (такими, например, как бюджет). Excel автоматически создает структуры (горизонтальную, вертикальную или смешанную), а также позволяет получать ее вручную. Однажды создав структуру, в дальнейшем вы можете разворачивать и сворачивать ее для отображения разных уровней детализации.

Автоматические промежуточные итоги

Excel позволяет автоматически вставлять (или удалять) формулы промежуточных итогов в таблицу, представленную в виде базы данных. Кроме того, с помощью Excel данные можно представлять структурировано, отображая промежуточные итоги или любой необходимый уровень детализации (рис. 2.11).



	A	B	C	D	E
1	**Месяц**	**Штат**	**Регион**	**Контракты**	**Продажи**
2	Январь	Калифорния	Запад	58	283 800
3	Январь	Вашингтон	Запад	35	507 200
4	Январь	Орегон	Запад	39	226 700
5			**Запад Итог**	132	1 017 700
6	Январь	Нью-Йорк	Восток	25	107 600
7	Январь	Нью-Джерси	Восток	47	391 600
8			**Восток Итог**	72	499 200
9	**Январь Итог**			204	1 516 900
10	Февраль	Калифорния	Запад	44	558 400
11	Февраль	Вашингтон	Запад	74	411 800
12	Февраль	Орегон	Запад	46	350 400
13			**Запад Итог**	164	1 320 600
14	Февраль	Нью-Йорк	Восток	52	233 800
15	Февраль	Нью-Джерси	Восток	29	154 200
16			**Восток Итог**	81	388 000
17	**Февраль Итог**			245	1 708 600
18	Март	Калифорния	Запад	30	353 100
19	Март	Вашингтон	Запад	57	258 400
20	Март	Орегон	Запад	44	532 100
21			**Запад Итог**	131	1 143 600
22	Март	Нью-Йорк	Восток	36	134 300
23	Март	Нью-Джерси	Восток	14	162 200
24			**Восток Итог**	50	296 500
25	**Март Итог**			181	1 440 100
26			**Общий итог**	630	4 665 600
27	**Общий итог**			630	4 665 600
28					
29					

Рис. 2.11. Excel позволяет автоматически вставлять итоги и промежуточные итоги

Analysis ToolPack

В надстройке Analysis ToolPack (Пакет анализа) содержится 19 специальных инструментов анализа (в основном, статистических по своей природе), а также ряд специализированных функций рабочих листов. Благодаря этим инструментам к программе Excel можно смело обращаться для проведения статистического анализа данных.

НОВИНКА

В предыдущих версиях программы отдельные статистические инструменты пакета анализа могли допустить ошибки в расчетах. Все недочеты в статистических инструментах полностью устранены в Excel 2003.

Сводные таблицы

Одним из самых мощных инструментов Excel являются *сводные таблицы*. Они позволяют сводить данные в виде удобной таблицы, которую можно приспособить для решения сложных задач (рис. 2.12). Для выполнения многих операций в сводной таблице достаточно перетащить данные с помощью мыши. Кроме того, объектами сводной таблицы можно управлять, используя возможности языка VBA. Данные этой таблицы импортируются из базы данных, которая находится на рабочем листе или загружается внешним образом, и хранятся в специальной кэш-памяти, позволяющей быстро выполнять пересчет данных после каждого изменения сводной таблицы.

ПЕРЕКРЕСТНАЯ ССЫЛКА

О том, как с помощью VBA управлять сводными таблицами, рассказывается в главе 17.

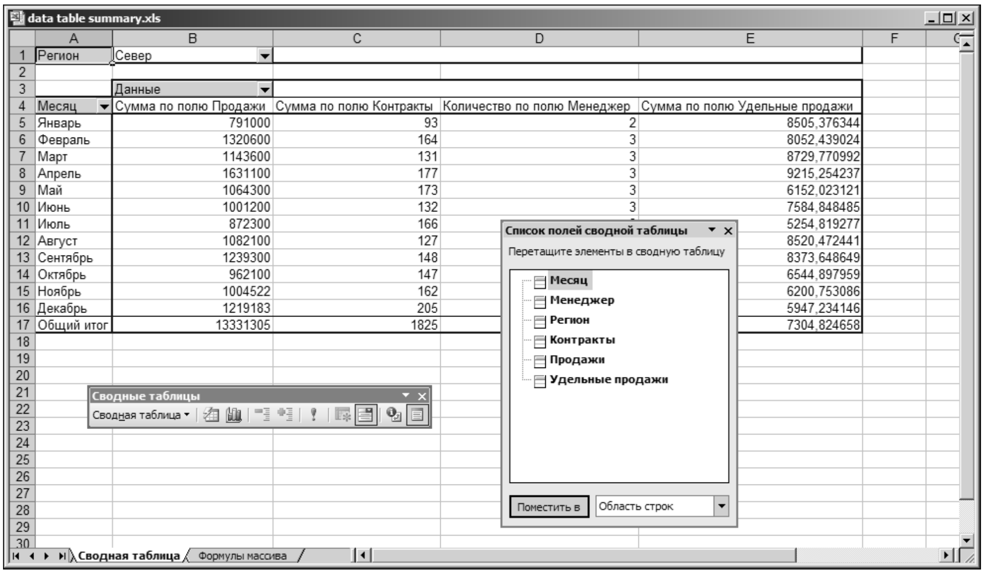


Рис. 2.12. Сводные таблицы имеют широкое применение в Excel

Поиск решения

Для решения специальных линейных и нелинейных задач в Excel применяется надстройка Поиск решения, которая использует структуры "что–если" для подбора данных в одних ячейках на основе ограничений, накладываемых на другие ячейки.

Надстройки

Надстройкой называется программа, внедренная в Excel с целью расширения функциональных возможностей последней. Чтобы подключить надстройку, выполните команду Сервис⇨Надстройки.

Кроме надстроек, которые поставляются вместе с Excel, существуют и другие, загружаемые с Web-узла компании Microsoft. Более того, на рынке также представлены надстройки сторонних производителей; эти надстройки можно или покупать, или загружать из Internet. Вы также имеете возможность создать свои собственные надстройки, о чем подробно рассказывается в главе 21.

Совместимость

Обычно файлы рабочих книг имеют особенности, характерные для той версии Excel, в которой эта книга создавалась. В программе Excel можно открывать файлы рабочих книг, созданные в предыдущих версиях. Наряду с этим открывать в ранних версиях Excel файлы, созданные в новых версиях, чаще всего не представляется возможным. Например, в Excel 97, Excel 2000–2002 используется один и тот же формат файлов, поэтому документы всех трех версий не вызывают проблем с совместимостью. Кроме того, в Excel, конечно же, можно сохранить рабочую книгу в одном из старых форматов.

В Excel существует возможность импорта самых разных файлов, созданных с помощью других процессоров электронных таблиц и приложений баз данных (более подробно об импорте данных рассказано в главе 4).

Отдельный аспект совместимости — это поддержка файлов предыдущих версий Excel. Здесь разработчиков могут подстерегать определенные сложности, которые обсуждаются в главе 26.

Параметры защиты

Excel предлагает несколько способов защиты данных рабочего листа. Например, вы можете защитить от изменения только формулы, структуру рабочей книги или VBA-код.

Защита формул

Во многих случаях нет необходимости защищать все данные рабочего листа. Достаточно не дать посторонним людям возможности изменять только формулы, с помощью которых выполняются вычисления.

1. Выберите ячейки с формулами, которые нужно защитить от изменения.
2. Выполните команду Формат⇨Ячейки. Щелкните на вкладке Защита диалогового окна Формат ячеек.
3. Снимите флажок опции Защищаемая ячейка и установите флажок опции Скрыть формулы.
4. Щелкните на кнопке ОК для закрытия диалогового окна Формат ячеек.
5. Выберите команду Сервис⇨Защита⇨Защитить лист. На экране появится диалоговое окно, которое показано на рис. 2.13. При использовании старых версий программы (до Excel 2002) диалоговое окно будет иметь несколько иной вид.

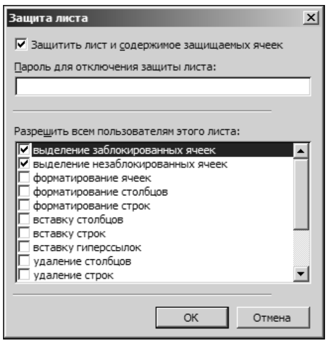


Рис. 2.13. Диалоговое окно Защита листа

6. В диалоговом окне Защита листа укажите необходимый пароль и щелкните на кнопке ОК.

По умолчанию все ячейки защищенные, хотя это никак не проявляется до назначения рабочему листу пароля.

В Excel 2003 настройки защиты стали разнообразнее. В диалоговом окне защиты рабочего листа вы можете точно указать, какие элементы следует защищать. Например, вы вправе разрешить пользователям сортировать защищенные данные или применять к ним автофильтр (в предыдущих версиях программы это сделать невозможно).

Если это необходимо, скройте формулы на рабочем листе. В результате при выделении ячеек с формулами последние не будут отображаться в строке формул. Для этого выставьте флажок Скрыть формулы на вкладке Защита диалогового окна Формат ячеек.

Защита структуры рабочей книги

При защите структуры рабочей книги вы не разрешаете добавлять или удалять из нее рабочие листы. Выполните команду Сервис⇨Защита⇨Защитить книгу. На экране появится диалоговое окно Защита книги, показанное на рис. 2.14. Убедитесь, что в нем выставлен флажок структуру. Если нужно запретить перемещение или изменение размера окна рабочей книги, то выставьте флажок окна.

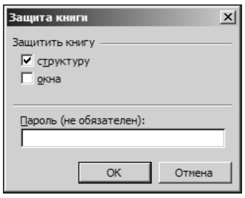


Рис. 2.14. Диалоговое окно Защита книги

Защита книги паролем

В некоторых случаях необходимо блокировать доступ сторонних пользователей к рабочей книге с помощью пароля. Для сохранения защищенной паролем рабочей книги выполните команду Файл⇨Сохранить как. В диалоговом окне Сохранение документа щелкните на кнопке Сервис и выберите команду Общие параметры. Вы увидите на экране диалоговое окно Параметры сохранения, показанное на рис. 2.15. Введите в нем пароль и щелкните на кнопке ОК. Для указания способа шифрования щелкните на кнопке Дополнительно.

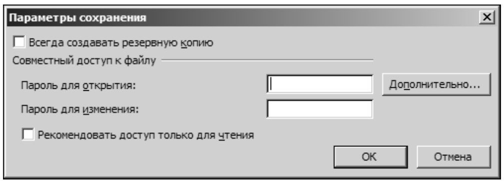


Рис. 2.15. Чтобы указать пароль, защищающий книгу от посторонних глаз, используйте диалоговое окно Параметры сохранения

Защита VBA-кода

С помощью пароля защищается не только вся рабочая книга или рабочий лист, но и VBA-код. Запустите редактор Visual Basic и выберите в окне Project защищаемый проект. Выполните команду Tools⇨*xxx Properties* (*xxx* соответствует имени проекта). На экране появится диалоговое окно свойств проекта.

Перейдите в этом окне на вкладку Protection (рис. 2.16). Выставьте флажок опции Lock Project for Viewing и введите пароль (дважды). Щелкните на кнопке ОК для сохранения проекта. После закрытия VBA-проекта и последующего его открытия вам необходимо будет ввести пароль.

Помните, что Excel не обладает абсолютной защитой от несанкционированного доступа к данным рабочего листа. Даже защита паролем не гарантирует, что ваши данные будут надежно спрятаны от постороннего взгляда. При желании любой опытный пользователь может взломать защиту с помощью специальных программ, которые бесплатно закачиваются из Internet.

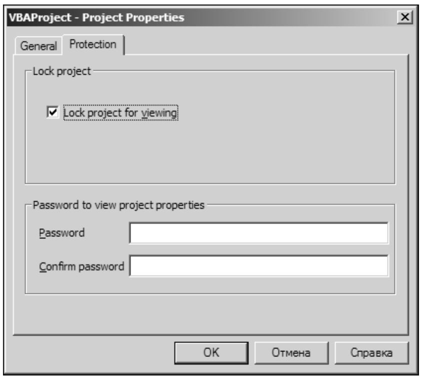


Рис. 2.16. Защита VBA-проекта с помощью диалогового окна Project Properties

Справочная система

Чуть не забыл о самом главном! О справочной системе. Она станет вашим лучшим помощником (после этой книги, конечно). Пользоваться ею очень просто. Введите вопрос в специальном поле в правой части строки меню и нажмите <Enter> (такое же поле вы найдете и в строке меню окна редактора VBA). В области задач будет отображен список разделов, соответствующих выбранной теме (рис. 2.17).

В Excel 2003 для получения справочных сведений лучше использовать область задач, а не отдельное окно. Однако при детальном изучении определенной темы вам придется открыть несколько окон справочной системы.

При подключении к Internet запрос отправляется на специальный сервер, в базе знаний которого выполняется поиск ответа на поставленный вопрос.

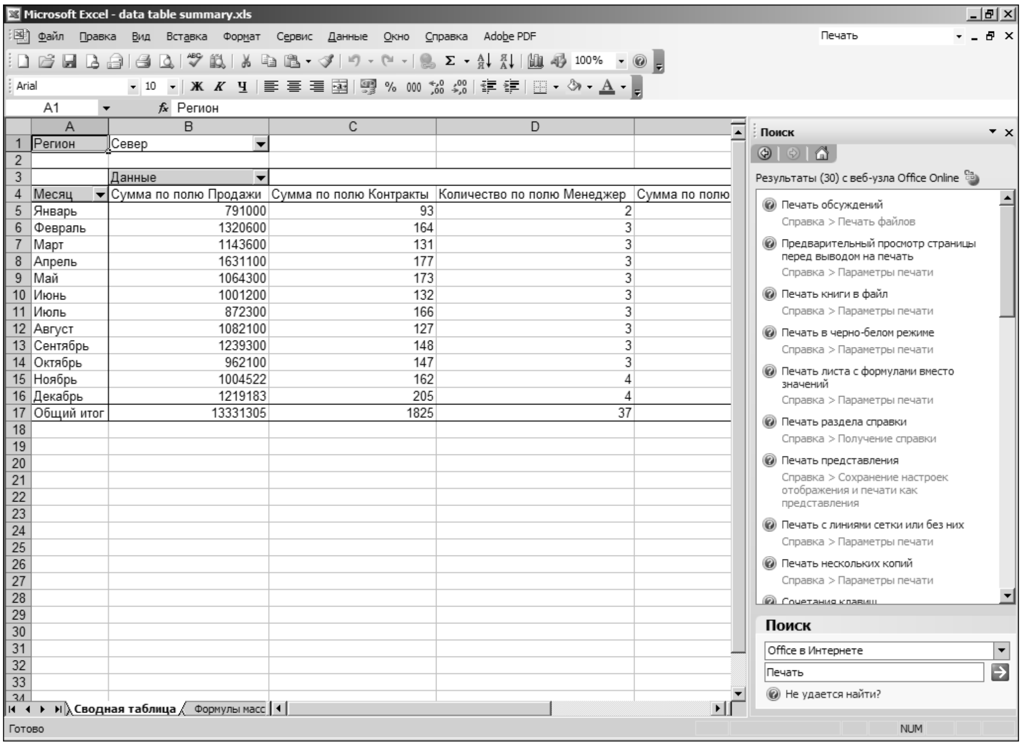


Рис. 2.17. Область задач содержит разделы, содержащие ответ на поставленный вопрос

Глава 3

Особенности использования формул

В ЭТОЙ ГЛАВЕ...

Формулы используются во многих сложных приложениях, созданных посредством электронных таблиц. Создание формул можно рассматривать как своего рода "программирование".

- Обзор формул Excel.
- Отличие абсолютных ссылок от относительных.
- Что такое имена и как они используются.
- Знакомство с формулами массивов.
- Подсчет и суммирование ячеек.
- Работа со значениями даты и времени.
- Создание мегаформул.

В этой главе описаны возможности Excel по управлению формулами и рассмотрены некоторые методы их применения.

Детально о создании и использовании формул рассказано в книге "Подробное руководство по созданию формул в Excel 2003", издательство "Диалектика".

О формулах

Формулы — это основа электронной таблицы. Если в таблице отсутствуют формулы, то она является просто статическим документом (который можно создать с помощью текстового процессора, обеспечивающего прекрасную поддержку статических таблиц).

Excel предлагает широкий набор встроенных функций, благодаря которым обеспечивается поддержка имен и даже формул массивов (специальный тип формул, способный творить чудеса).

Ниже приведены элементы, которые представляют часть введенной в ячейку формулы:

- операторы — + (сложение) и * (умножение);
- ссылки на ячейки (в том числе имена ячеек и диапазонов);
- значения или строки;
- функции рабочих листов (например, `СУММ` или `СРЗНАЧ`).

Формула может содержать до 1024-х символов. После введения формулы в ячейку последняя отображает результат выполнения формулы. Впрочем, если ячейку активизировать, то в строке формул появится сама формула.

Вычисление значений формул

Вы, возможно, заметили, что формулы в рабочем листе вычисляются сразу. И если изменить ячейку, которая используется в формуле, то результат будет пересчитан без вашего участия. Это происходит в тех случаях, когда параметр Вычисления установлен в значение автоматически. Режим, задаваемый этим значением, выставлен по умолчанию. Вычисления в рабочем листе, которые выполняются в этом режиме, характеризуются следующими правилами.

- Когда вы вносите изменения (например, вводите или редактируете данные или формулы), Excel немедленно вычисляет значения формул уже с учетом новых или отредактированных данных.
- Иногда процесс длительного вычисления формул Excel временно приостанавливает, чтобы вы могли выполнить другие задачи, связанные с управлением рабочим листом. Когда же вы оканчиваете свои действия, вычисление возобновляется.
- Формулы вычисляются в естественном порядке. Другими словами, если формула в ячейке `D12` зависит от результата вычисления формулы в ячейке `D11`, то сначала вычисляется значение в ячейке `D11`, а только потом — в ячейке `D12`.

Впрочем, иногда контроль над вычислением значений формул в Excel вам, возможно, придется брать на себя. Например, создавая рабочий лист с тысячами сложных формул, вы заметите, что при вычислении их значений скорость обработки данных резко уменьшается. В таком случае рекомендуется установить ручной режим вычисления (на вкладке Вычисления диалогового окна Параметры).

Если при работе в ручном режиме вычисления рабочий лист содержит непросчитанную формулу, то в строке состояния будет отображено сообщение Вычислить. Для пересчета значений формул можно использовать следующие клавиши.

- При нажатии клавиши <F9> вычисляются значения формул во всех открытых рабочих книгах.
- При нажатии комбинации клавиш <Shift+F9> вычисляются значения формул только в активном рабочем листе. В других рабочих листах этой же рабочей книги вычисления не выполняются.
- Комбинация клавиш <Ctrl+Shift+F9> приводит к пересчету абсолютно всего. Эта комбинация клавиш является незадокументированной. Используйте ее, если Excel по какой-либо причине явно рассчитывает данные неправильно, или существует необходимость выполнить пересчет формул, в которых применяются пользовательские функции, созданные на языке VBA.

НА ЗАМЕТКУ

В Excel режим вычисления невозможно изменить только для одной (текущей) рабочей таблицы. Изменение этого режима затрагивает все открытые рабочие книги, а не только активную.

Ссылки на ячейки и диапазоны

В большинстве формул присутствуют ссылки не более чем на одну ячейку. Такие ссылки задаются с помощью адреса или имени (если оно задано), определяющих как ячейку, так и диапазон ячеек. Ссылки на ячейки бывают четырех типов.

- *Относительная.* Ссылка является полностью относительной. Когда формула копируется, то ссылка изменяется в соответствии с новым местоположением формулы (например: `A1`).

- *Абсолютная*. Ссылка является полностью абсолютной. Когда формула копируется, ссылка не меняется (например: `$A$1`).
- *Абсолютная строка*. Ссылка является частично абсолютной. Когда формула копируется, то та часть ссылки, которая указывает столбец, меняется в соответствии с новым местоположением формулы, а строчная часть ссылки остается неизменной (например: `A$1`).
- *Абсолютный столбец*. Ссылка является частично абсолютной. Когда формула копируется, то строчная часть ссылки меняется в соответствии с новым местоположением формулы, а та часть ссылки, которая указывает столбец, остается неизменной (например: `$A1`).

По умолчанию все ссылки на ячейки и диапазоны являются относительными. Чтобы изменить тип ссылки, следует вручную добавить к ней знаки доллара. Можно сделать и по-другому: когда ячейка редактируется в строке формул, переместите курсор к нужному адресу, а затем нажимайте клавишу <F4> до тех пор, пока методом подбора не получите необходимый тип ссылки.

Неотносительные ссылки

Думая об этом, вы поймете, что единственная причина, по которой когда-либо придется изменить тип ссылки — это необходимость копирования формулы (рис. 3.1). Пусть в ячейке `C4` находится следующая формула.

```
=C$3*$B4
```

Данная формула вычисляет площадь прямоугольника для различных значений его ширины (перечисленных в столбце `B`) и длины (перечисленных в строке `3`). После ввода формулу скопировали вниз, в ячейку `C8`, а затем вправо, в ячейку `F8`. Поскольку в формуле используются ссылки (одна с абсолютной строкой `3`, другая с абсолютным столбцом `B`, остальные части этих ссылок являются относительными), каждая скопированная формула все равно будет давать правильный результат. Если в формуле применяются только относительные ссылки, то в результате ее копирования все ссылки изменятся, что приведет к неправильным результатам.

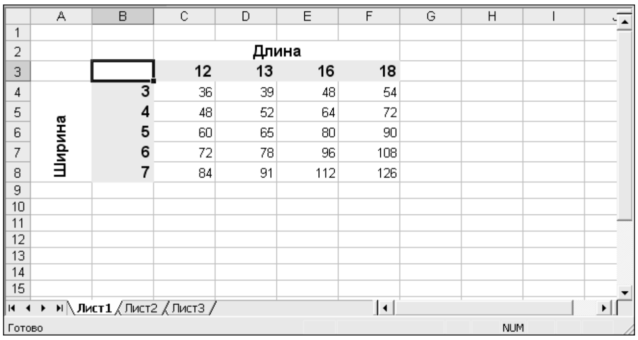


Рис. 3.1. Пример формулы с неотносительными ссылками

О ссылках R1C1

Как правило, в Excel используется формат записи ссылок A1. Каждый адрес ячейки, отображаемый в таком формате, состоит из буквы, которая обозначает столбец, и числа, которое соответствует строке. Впрочем, в Excel также поддерживается формат записи ссылок R1C1. (Здесь R означает *row*, т.е. "строка", а C — *column*, т.е. "столбец"). В этом формате ячейка `A1` обозначается как R1C1, `A2` — соответственно, как R2C1 и т.д.

Чтобы перейти к формату R1C1, выберите команду Сервис⇨Параметры, щелкните на вкладке Общие и установите флажок Стиль ссылок R1C1. После этого вы обнаружите, что все буквы столбцов заменены на числа. Более того, соответствующим образом в созданных ранее формулах изменяются все ссылки на ячейки и диапазоны.

В табл. 3.1 приведены примеры формул, использующих стандартный формат записи и формат R1C1. Предполагается, что каждая из этих формул находится в ячейке `B1` (также известной как `R1C2`).

Таблица 3.1. Сравнение простых формул, выведенных в одном из двух форматов записи

Стандартный	R1C1
`=A1+1`	`=RC[-1]+1`
`=$A$1+1`	`=R1C1+1`
`=$A1+1`	`=RC1+1`
`=A$1+1`	`=R1C[-1]+1`
`=СУММ(A1:A10)`	`=СУММ(RC[-1]:R[9]C[-1])`
`СУММ($A$1:$A$10)`	`=СУММ(R1C1:R10C1)`

Если формат записи ссылок R1C1 вы считаете запутанным, то вы не одиноки в своих умозаключениях. Он применяется для введения абсолютных ссылок, но когда используются относительные ссылки, то от квадратных скобок легко сойти с ума.

Числа в квадратных скобках обозначают относительное местоположение ссылок. Например, ссылка R[-5]C[-3] указывает на ячейку, которая находится на пять строк выше и на три столбца левее той ячейки, в которой расположена текущая ссылка. С другой стороны, ссылка R[5]C[3] обозначает ячейку, которая расположена на пять строк ниже и три столбца правее текущей. Если квадратных скобок нет, то это указывает на ту же самую строку или тот же самый столбец. Например, R[5]C указывает на ячейку, расположенную на пять строк ниже в текущем столбце.

Скорее всего, формат R1C1 не станет для вас используемым по умолчанию, однако он *все же* вам пригодится. С его помощью легко отыскать формулу с ошибкой. Если вами используется формат R1C1, то любые копии одной и той же формулы будут одинаковыми. Это относится ко всем типам применяемых вами ссылок на ячейки (относительных, абсолютных или смешанных). Можете перейти в режим R1C1 и проверить скопированные формулы. И если какая-либо из них отличается от остальных, то, скорее всего, она и является неправильной.

Кроме того, если вы создаете код VBA для получения формул рабочих листов, то, возможно, предпочтете формат R1C1.

Ссылки на другие листы или рабочие книги

Ячейки и диапазоны, на которые задаются ссылки в формуле, не обязательно должны существовать в том же листе, что и сама формула. Если в формуле требуется указать ячейку из другого листа, то перед ссылкой на саму ячейку введите имя этого листа, а после имени — восклицательный знак. Ниже приведен пример формулы со ссылкой на ячейку, расположенную в другом рабочем листе.

```
=Лист2!A1+1
```

Кроме того, можно создавать формулы со ссылками на те ячейки, которые расположены в другой рабочей книге. Для этого перед ссылкой на саму ячейку введите имя рабочей книги (в квадратных скобках), имя рабочего листа и восклицательный знак.

```
=[Бюджет.xls]Лист1!A1+1
```

Если в имени рабочей книги, используемом в ссылке, содержатся пробелы, то его (вместе с именем рабочего листа) необходимо заключить в одинарные кавычки.

```
='[Бюджет на 2002 год]Лист1'!A1+1
```

Указанная в ссылке рабочая книга может быть закрыта, тогда в ссылке следует указать полный путь к этой книге.

```
='C:\MSOffice\Excel\[Бюджет на 2002 год]Лист1'!A1+1
```

В формулах ссылки на рабочие книги указываются в виде пути. Однако вы вправе обратиться к методу указания мышью. Для этого исходный файл должен быть открытым. В данном случае создаются абсолютные ссылки на ячейки (если вы собираетесь копировать формулу в другие ячейки, то ссылки обязательно измените на относительные).

Использование ссылок для восстановления данных в поврежденном файле

В Excel 2002 появилась новая функция обнаружения и восстановления, с помощью которой можно восстановить поврежденный или испорченный файл. Если она не помогает устранить проблему (или вы пользуетесь более ранней версией Excel), то воспользуйтесь описанным ниже приемом.

Если не удается загрузить поврежденную рабочую книгу Excel, напишите формулу со ссылкой, чтобы восстановить все или часть данных (но только не формулы). Дело в том, что исходный файл, указанный в формуле со ссылкой, открывать нет необходимости. Если испорченный файл называется, например, `Badfile.xls`, то для восстановления данных листа `Лист1` поврежденного файла откройте пустую рабочую книгу и на ее листе `Листе1` введите в ячейке `A1` следующую формулу.

```
='C:\Files\[Badfile.xls]Лист1'!A1
```

Затем в новой рабочей книге скопируйте эту формулу вниз и вправо, чтобы восстановить как можно больше информации. Впрочем, существует способ и получше — регулярно выполнять резервное копирование всех важных файлов.

Работа со ссылками может показаться вам сложной операцией. Например, если для создания резервной копии исходной рабочей книги вы используете команду Файл⇨Сохранить как, то формулы со ссылками автоматически изменяются, чтобы обращаться к указанному файлу (но имеющему новое имя). Существует еще одна возможность нарушить ссылки: переименуйте исходную рабочую книгу, когда не открыта зависящая от нее рабочая книга.

Использование имен

Одна из самых существенных возможностей программы Excel — это назначение самым разным элементам содержательных имен. Имена можно присваивать ячейкам, диапазонам ячеек, строкам, столбцам, диаграммам и другим объектам. Преимущество, которым обладает только Excel, позволяет присваивать имена тем значениям или формулам, которые даже не отображаются в ячейках рабочего листа (смотрите далее в этой главе раздел “Присвоение имен константам”).

Присвоение имен ячейкам и диапазонам

Имена ячеек и диапазонов можно создавать с помощью команды Вставка⇨Имя⇨Присвоить (или комбинации клавиш <Ctrl+F3>). Впрочем, создавать имена можно еще быстрее (воспользуйтесь полем имен — раскрывающимся списком, который расположен в левой части строки формул). Вам достаточно выбрать одну ячейку или диапазон ячеек, ввести необходимое имя в поле имен, а затем нажать <Enter>.

Имена ячеек или диапазонов можно создавать автоматически, на основе заголовков строк и столбцов рабочего листа. Для этого выполните команду Вставка⇨Имя⇨Создать. Например, на рис. 3.2 показано, что диапазон `C4:F4` получил название *Север*, `C5:F5` — название *Юг* и т.д. Что касается вертикальных диапазонов, то `C4:C7` назван *Квартал1*, `D4:D7` — *Квартал2* и т.д.

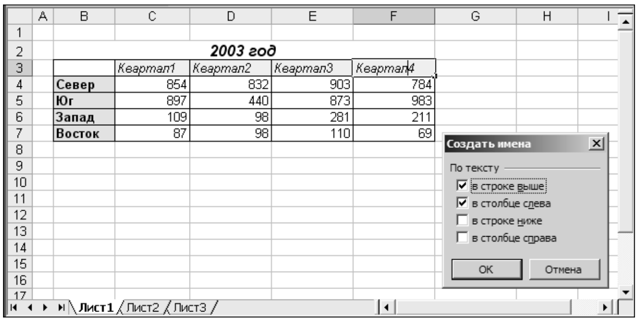


Рис. 3.2. В рабочих листах Excel можно создавать описательные имена

Использование имен особенно эффективно при написании кода VBA, в котором применяются ссылки на отдельные ячейки или диапазоны. Почему же так важны имена? Ответ заключается в следующем: если ячейку или диапазон, на которые ссылается оператор VBA, вы переместите в другое место, то в VBA-коде эти ссылки автоматически обновляться не будут. Например, если в VBA-коде значение записывается в ячейку `C4`, заданную как `Range("C4")`, то после вставки новой строки над этой ячейкой или нового столбца слева от нее данные будут записываться не в требуемую ячейку. Чтобы не возникало подобных проблем, применяйте ссылки на именованные ячейки, например, `Range("InterestRate")`.

Использование имен существующих ссылок

Когда ячейке или диапазону ячеек назначается имя, то Excel автоматически не использует его вместо ссылок на ячейку или диапазон, которые уже содержатся в формулах. Предположим, что в ячейке `F10` находится формула.

```
=A1-A2
```

Если для `A1` вы зададите имя *Доходы*, а для `A2` — имя *Расходы*, то формула автоматически не будет превращена в `=Доходы-Расходы`. Впрочем, заменить именами ссылки на ячейки и диапазоны несложно. Вначале выделите тот диапазон, в котором необходимо сделать изменения. Затем выполните команду Вставка⇨Имя⇨Применить. В появившемся диалоговом окне выделите имена, которые следует применить при замене, а затем щелкните на кнопке OK. В результате все ссылки на ячейки и диапазоны, имеющие имена, будут заменены ссылками на имена.

К сожалению, способа “отключения” имен не существует. Другими словами, если в формуле используется имя, то его нельзя преобразовать в явную ссылку на ячейку или диапазон. Хуже того, если удалить имя, используемое в формуле, то программа не вернется к обычному способу адресации ячейки или диапазона — она выведет сообщение об ошибке `#ИМЯ?`.

Скрытые имена

Отдельные макросы и надстройки Excel создают скрытые имена. Так называются имена, которые в рабочей книге содержатся, но в диалоговом окне Применение имени их не видно. Например, большое количество скрытых имен создается надстройкой Поиск решения. Эти скрытые имена можно игнорировать. Впрочем, иногда они создают проблему. При копировании листа в другую рабочую книгу скрытые имена также копируются, кроме того, они могут создать ссылку, которую трудно обнаружить.

Для удаления из рабочей книги всех скрытых имен используйте следующую процедуру VBA.

```
Sub DeleteHiddenNames()
Dim n As Name
Dim Count As Integer
    For Each n In ActiveWorkbook.Names
        If Not n.Visible Then
            n.Delete
            Count = Count + 1
        End If
    Next n
    MsgBox "Скрытые имена в количестве " & Count & " удалены"
End Sub
```

В состав надстройки Power Utility Pak (находится на прилагаемом компакт-диске) входит утилита, которая в выделенной области “просматривает” все формулы и автоматически заменяет имена на соответствующие ссылки.

Пересечение имен

В программе Excel существует специальный оператор (*оператор пересечения*). Он вступает в действие, когда возникает необходимость в управлении несколькими диапазонами ячеек. Этим оператором является символ пробела. Используя оператор пересечения вместе с именами, легко создавать достаточно содержательные формулы. Для наглядного примера обратитесь к рис. 3.2. Если в ячейку ввести следующую формулу

```
=Квартал2 Юг
```

то результатом будет 440 — пересечение диапазонов *Квартал2* и *Юг*. Чтобы получить итог по западному региону (`Запад`), можете использовать такую формулу.

```
=СУММ(Запад)
```

Ссылки на “естественном языке”

Начиная с Excel 97, можно вводить формулы на “естественном языке”, в котором используются заголовки строк и столбцов. Не следует специально определять эти имена — Excel их вычислит автоматически. Такие псевдоимена объединяются с помощью оператора пересечения (т.е. символа пробела). Например, вы создаете формулу (которая дословно означает “продажи за январь”).

```
=Январь Продажи
```

В результате должно отображаться значение, которое находится на пересечении столбца с заголовком `Продажи` и строки с заголовком `Январь`.

Данная возможность достаточно удобна, однако советуем остерегаться ее использования. Псевдоимена ненадежны и трудны для документирования, кроме того, их нельзя использовать в коде VBA. Когда эта функция только лишь появилась, компания Microsoft оценила ее как существенное облегчение для пользователей. Однако в настоящее время такая возможность по умолчанию является отключенной. Для ее включения необходимо установить соответствующий флажок на вкладке Вычисления диалогового окна Параметры.

Присвоение имен столбцам и строкам

Excel предоставляет возможность присваивать имена отдельным строкам и столбцам. В последнем примере имя *Квартал1* присвоено диапазону `C4:C7`. Однако это имя можно присвоить всему столбцу `C`, имя *Квартал2* — столбцу `D` и т.д. То же самое можно сделать и "по горизонтали" — имя *Север* поставить в соответствие строке `4`, а *Юг* — строке `5` и т.д.

Оператор пересечения используется в данном случае так же, как и раньше, но теперь в рабочий лист можно добавлять другие регионы или кварталы, не меняя при этом уже существующих имен.

Присваивая имена столбцам и строкам, проверяйте, чтобы в самих строках и столбцах не было лишних данных. Помните, если вы, например, вставите значение в ячейку `C7`, то оно попадет в диапазон *Квартал1.*

Задание области действия

Областью действия именованной ячейки или диапазона обычно является рабочая книга — другими словами, имя можно использовать в любом рабочем листе рабочей книги.

Существует и другой вариант — создание имен, областью действия которых является рабочий лист. Для этого перед самим именем должно стоять имя рабочего листа, а затем — восклицательный знак (например, *Лист1!Продажи*). Если имя применяется в том листе, для которого оно предназначено, то при обращении к нему упоминание о рабочем листе можно опускать. Что касается диалогового окна Присвоение имени, то в нем имена с областью действия на уровне рабочего листа появятся только тогда, когда лист, на котором они определены, является активным. Впрочем, обращаться можно и к тем именам уровня рабочего листа, которые определены в другом листе. В таком случае перед выбранным именем следует добавлять имя рабочего листа.

Если вы решили совместно использовать имена с областью действия на уровне рабочей книги и на уровне рабочего листа, то обязательно удостоверьтесь, что знаете, как они работают, иначе вас могут подстерегать неожиданные сюрпризы.

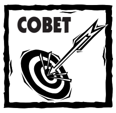
Присвоение имен константам

Каждый опытный пользователь Excel знает, как создавать имена ячеек и диапазонов (хотя не все пользователи Excel применяют это в своей практике). Заметим, что, кроме всего прочего, имена можно применять для обращения к значениям, которые не встречаются в рабочем листе (т.е. для обращения к *константам*).

Предположим, в нескольких формулах рабочего листа используется конкретное значение процентной ставки. Вы можете вставить это значение в ячейку и дать ему имя (например, *Ставка*), чтобы впоследствии применять его в своих формулах. Например, обратимся к нему в следующей формуле.

```
=Ставка*A3
```

Существует и другой способ — открыть диалоговое окно Присвоение имени и ввести значение процентной ставки в поле Формула (рис. 3.3). Затем назначенное процентной ставке имя можно использовать в формулах так, как если бы оно хранилось в ячейке. В случае изменения процентной ставки всего лишь измените определение имени *Ставка* — все ячейки, в которых оно содержится, будут обновлены.



Данный прием также используется для текстовых данных. Например, вы можете определить имя *МКП* для значения Международная корпорация простаков. Если впоследствии ввести в ячейку формулу **=МКП**, то в ней будет отображено полное название.

Присвоение имен формулам

Имена можно присваивать не только ячейкам, диапазонам и константам. Вы также вправе ввести формулу в поле Формула диалогового окна Присвоение имени и создать таким образом именованную формулу. Введенная вами формула имеет относительные ссылки, если рассматривать ее с точки зрения ячейки, в которой она находится. Впрочем, если при создании формулы для указания ячеек использовалась мышь, то ссылки будут абсолютными.

На рис. 3.4 показана формула (`=A1^B1`), введенная в поле Формула диалогового окна Присвоение имени. В этом случае активна ячейка `C1`, поэтому формула обращается к двум ячейкам, которые находятся левее (обратите внимание, что ссылки на ячейки являются относительными). Если после определения имени ввести в какую-либо ячейку формулу `=Степень`, то значение, находящееся на две ячейки левее, будет возведено в степень, указанную в ячейке слева. Например, если в ячейках `B10` и `C10` находятся, соответственно, 3 и 4, то ввод следующей формулы в ячейку `D10` приведет к выводу значения, равного 81 (3 в 4-й степени).

```
=Степень
```

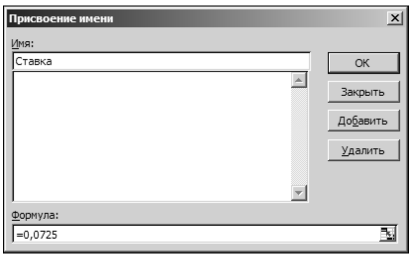


Рис. 3.3. Excel предоставляет возможность присваивать имена тем константам, которые не встречаются в ячейках рабочего листа

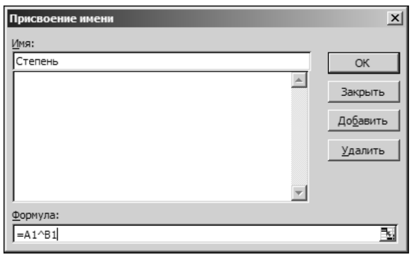


Рис. 3.4. Имя можно присвоить формуле, которая не встретится ни в одной ячейке рабочего листа

Открыв после создания именованной формулы диалоговое окно Присвоение имени, вы обнаружите, что в поле Формула отображается формула, которая является относительной для активной ячейки. Например, если активна ячейка `D32`, то в поле Формула появится следующая формула.

```
=Лист1!B32^Лист1!C32
```

Обратите внимание, что в ссылки добавлено имя рабочего листа. Таким образом, если именованную формулу использовать за пределами рабочего листа, в котором она определена, то ее значения могут быть неправильными. Если же требуется применить

именованную формулу в ином листе, чем `Лист1`, то из формулы придется удалить все ссылки на лист (однако сохранив восклицательные знаки).

```
=!A1^!B1
```

Разобравшись с именованными формулами, вы, возможно, найдете для них новое применение. Неоспоримое преимущество наблюдается в том случае, если в формуле необходимо провести изменения. Вы можете с помощью окна Присвоение имени изменить определение формулы, а не редактировать каждый ее экземпляр на рабочем листе.

Прилагаемый к книге компакт-диск содержит несколько примеров рабочих книг, в которых применяются именованные формулы.

Что представляют собой имена ячеек и диапазонов

Опытные пользователи Excel часто говорят об *именованных диапазонах* и *именованных ячейках*. В данной главе мы часто пользуемся этими терминами. Однако употребляемая нами терминология не совсем точна.

Итак, откроем секрет, чем же в действительности являются имена.

Создавая в Excel имя для ячейки или диапазона ячеек, вы на самом деле создаете именованную формулу, т.е. формулу, которой нет в ячейке. Эти именованные формулы находятся не в ячейках, а в буфере Excel.

Когда вы работаете в диалоговым окне Присвоение имени, то в поле Формула отображается формула, а в другом поле (Имя) вводится имя данной формулы. Примечательно, что содержимое поля Формула всегда начинается со знака равенства — он и делает содержимое поля формулой.

Если вы будете помнить этот “секрет”, то вам несложно будет разобраться в действиях, происходящих при создании и использовании имен в рабочих книгах.

Присвоение имен объектам

Кроме присвоения имен ячейкам и диапазонам, вы также можете давать содержательные имена таким объектам, как, например, диаграммы и фигуры. К ним будет проще обращаться, особенно в коде VBA.

Впрочем, если вы думаете, что имена объектам присваиваются с помощью той же команды Вставка⇨Имя⇨Присвоить, то ошибаетесь (она применяется только для именования ячеек и диапазонов). Единственный способ изменить имя объекта, не являющегося диапазоном, — это использование поля имен. Выделите сам объект, затем введите в данном поле новое имя и нажмите клавишу <Enter>.

Чтобы введенное вами имя не исчезло, недостаточно, введя в поле имен новое значение, щелкнуть в произвольной области рабочей книги. *Обязательно* нажмите клавишу <Enter>.

Ошибки использования формул

Нередко бывает так, что, введя формулу, вы в ответ получаете значение, которое сообщает об ошибке. Формулы возвращают такое значение, если в ячейке, на которую они ссылаются, находится ошибочное значение. Это называется “цепной реакцией” — единственное ошибочное значение вызывает образование целого ряда ошибочных значений в других ячейках, в которых содержатся формулы, зависящие от ячейки

с исходным значением. Инструменты, которые помогают отслеживать источники ошибок в формулах, находятся на панели инструментов Зависимости.

В табл. 3.2 перечислены значения-сообщения об ошибках, которые могут появиться в ячейках с формулами.

Таблица 3.2. Значения Excel, которые сообщают об ошибках

Значение	Описание
`#ДЕЛ/0!`	В формуле производится попытка поделить на нуль (операция, запрещенная законами математики). Подобная ошибка появляется и в том случае, когда в формуле осуществляется деление на содержимое пустой ячейки
`#Н/Д`	Формула ссылается (прямо или косвенно) на ячейку, в которой используется такая функция рабочего листа, как `НД`. Применение этой функции указывает на то, что данные недоступны. Кроме того, значение `#Н/Д` возвращается функцией `ПРОСМОТР`, которая не смогла найти значение
`#ИМЯ?`	В формуле используется имя, которое программа Excel не признает. Это случается, если имя, определенное в формуле, удалено, или в тексте не совпадает количество открывающих и закрывающих кавычек
`#ПУСТО!`	В формуле применяется пересечение двух диапазонов, которые на самом деле не пересекаются (об этом рассказывается далее в настоящей главе)
`#ЧИСЛО!`	Проблема возникла со значением (например, используется отрицательное число тогда, когда ожидается положительное)
`#ССЫЛ!`	В формуле определена ссылка на недопустимую ячейку. Это может произойти, если ячейка удалена из рабочего листа
`#ЗНАЧ!`	В формуле присутствует аргумент или операнд неправильного типа. Операнд — это значение или ссылка на ячейку, используемые формулой для вычисления результата. Кроме того, такая ошибка проявляется, если в формуле применена пользовательская VBA-функция с собственной ошибкой

Формулы массивов

Массив — это коллекция ячеек или значений, которой управляют как единым целым. *Формулой массива* является формула специального вида, которая обрабатывает массивы данных. Результатом выполнения формулы массива может быть как единичный результат, так и набор значений, причем каждое из них помещается в отдельную ячейку (Excel допускает расположение в одной ячейке только одного значения).

Например, если вы умножаете массив 1×5 на массив 1×5, то в результате получаете массив 1×5. Другими словами, результат выполнения подобной операции занимает пять ячеек рабочего листа; каждый элемент первого массива умножается на соответствующий элемент второго массива. Вы получите пять новых значений, каждое из которых будет занимать собственную ячейку. Следующая формула массива умножает значения массива `A1:A5` на соответствующие значения массива `B1:B5`. Такая формула должна вводиться одновременно в пять ячеек.

```
=A1:A5*B1:B5
```

Формула массива вводится нажатием комбинации клавиш <Ctrl+Shift+Enter>. Напоминанием о том, что в строке формул содержится формула массива, служат фигурные скобки ({}), в которые она заключена. Не вводите эти скобки вручную!

Пример формулы массива

В Excel с помощью формул массивов можно выполнять отдельные операции над каждой ячейкой диапазона, причем во многом таким же образом, как и посредством циклических структур языка программирования. Если вам еще не приходилось использовать формулы массивов, то внимательно рассмотрите описанный далее пример.

На рис. 3.5 представлена электронная таблица с текстом в ячейках `A1:A5`. Цель данного упражнения состоит в том, чтобы создать *единственную формулу*, которая возвратит сумму, равную общему количеству символов в этом диапазоне. Если не требовать выполнения этой задачи с помощью единственность формулы, то можно создать формулу с функцией `ДЛСТР`, скопированной во все ячейки столбца `B`, а затем с помощью функции `СУММ` сложить результаты промежуточных формул.

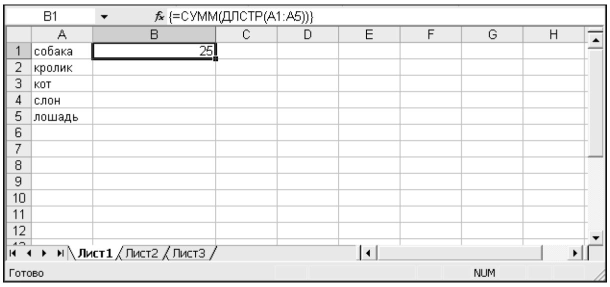


Рис. 3.5. В ячейке `B1` находится формула массива, которая возвращает общее число символов, содержащихся в диапазоне `A1:A5`

Вы сможете убедиться в том, что формула массива может занимать более одной ячейки, если создадите рабочий лист, показанный на рис. 3.5, а затем выполните следующие действия.

1. Выделите диапазон `B1:B5`.
2. Введите следующую формулу.

   ```
   =ДЛСТР(A1:A5)
   ```

3. Нажмите комбинацию клавиш <Ctrl+Shift+Enter>.

Эти действия выполняются для введения единственной формулы в пять ячеек. Затем введите формулу `СУММ`, которая складывает длины значений из ячеек `B1:B5`, и тогда вы увидите, что в ячейках `A1:A5` находится всего 25 символов.

Главное в этом примере то, что полученные пять элементов массива в ячейках `B1:B5` *отображать* не обязательно, так как массив может храниться в памяти. Помня об этом, вы вправе в любую пустую ячейку ввести следующую формулу (обязательно с помощью комбинации клавиш <Ctrl+Shift+Enter>).

```
=СУММ(ДЛСТР(A1:A5))
```

Отображенная формула будет заключена в фигурные скобки.

```
{=СУММ(ДЛСТР(A1:A5))}
```

Указанная формула создает (в памяти) массив из пяти элементов, которыми являются значения длины каждой строки массива, расположенного в ячейках `A1:A5`. Этот массив значений длин используется в качестве аргумента функции `СУММ` — в результате формула возвращает значение 25.

Календарь в виде формулы массива

На рис. 3.6 показан рабочий лист, который отображает календарь для любого месяца. Хотите — верьте, а хотите — нет, но календарь создается с помощью единственной формулы массива, которая занимает 42-е ячейки.

На компакт-диске, прилагаемом к настоящей книге, представлена рабочая книга с примером календаря, а также несколько других примеров формул массивов.



Рис. 3.6. Единственная формула массива — все, что необходимо для создания календаря на любой месяц года

Достоинства и недостатки формул массивов

Ниже перечислены преимущества формул массивов в сравнении с формулами для одной ячейки.

- Зачастую требуют меньше памяти.
- Позволяют выполнять вычисления намного эффективнее.
- Не требуют наличия промежуточных формул.
- Предоставляют возможность выполнения операций, которые реализовать по-другому трудно или вообще невозможно.

Впрочем, у формул массивов имеются и свои недостатки.

- Некоторые формулы существенно замедляют пересчет электронной таблицы.
- Они мешают другим пользователям разобраться в созданной вами таблице.
- Помните, что формула массива вводится с помощью специальной комбинации клавиш <Ctrl+Shift+Enter>.

Подсчет и суммирование

Анализ данных, появляющихся в группах новостей Internet, занимает немало времени. Многие вопросы из таких групп новостей относятся к подсчету и суммированию разного рода информации. Мы ответим на большинство из них, приведя в качестве примера удобные формулы, которые подсчитывают различные данные рабочего листа.

Использование функций СЧЕТЕСЛИ и СУММЕСЛИ

Такие функции Excel, как `СУММ`, `СЧЕТ`, `СЧЕТЗ` и `СЧИТАТЬПУСТОТЫ`, довольно просты в использовании, поэтому мы не будем на них останавливаться и сразу перейдем к рассмотрению более полезных функций `СЧЕТЕСЛИ` и `СУММЕСЛИ`.

- Функция `СЧЕТЕСЛИ` принимает два аргумента:
  - диапазон ячеек, содержащий те данные, которые необходимо посчитать;
  - критерии, на основе которых ячейка должна или не должна учитываться при подсчете.
- Функция `СУММЕСЛИ` принимает три аргумента:
  - проверяемый диапазон;
  - критерии, на основе которых ячейка должна или не должна учитываться при подсчете;
  - также диапазон, содержащий суммируемые данные.

В табл. 3.3 представлены случаи использования функции `СЧЕТЕСЛИ`. Предполагается, что у вас есть диапазон ячеек с именем *Данные* (вам потребуется вместо этого имени подставить в формулы имя настоящего диапазона или ячейки). Кроме того, не забывайте, что вторым аргументом функции `СЧЕТЕСЛИ` может быть ссылка на ячейку, содержащую критерии поиска.

Таблица 3.3. Примеры наиболее частого использования функции СЧЕТЕСЛИ

Формула	Возвращаемое значение
`=СЧЕТЕСЛИ(Данные;12)`	Количество ячеек, которые содержат значение, равное 12
`=СЧЕТЕСЛИ(Данные;1)+ СЧЕТЕСЛИ(Данные;12)`	Количество ячеек, которые содержат 1 или 12
`=COUNTIF(Данные;"<0")`	Количество ячеек, которые содержат отрицательное число
`=СЧЕТЕСЛИ(Данные;"<>0")`	Количество ненулевых значений
`=СЧЕТЕСЛИ(Данные;">=1")- СЧЕТЕСЛИ(Данные;">10")`	Количество ячеек, которые содержат значение от 1 до 10
`=СЧЕТЕСЛИ(Данные;"yes")`	Количество ячеек, которые содержат слово *yes* (регистр не учитывается)
`=СЧЕТЕСЛИ(Данные;"*")`	Количество ячеек, которые содержат любой текст
`=СЧЕТЕСЛИ(Данные;"*s*")`	Количество ячеек, которые содержат букву *s* (регистр не учитывается)
`=СЧЕТЕСЛИ(Данные;"???")`	Количество слов из трех букв

Подсчет и суммирование с помощью формул массивов

Если ни один из стандартных приемов, используемых для подсчета, не подходит, то создайте формулу массива (смотрите ранее в этой главе раздел “Формулы массивов”). Не забывайте о том, что, введя формулу массива, необходимо нажать комбинацию клавиш <Ctrl+Shift+Enter>.

Чтобы подсчитать количество числовых значений (исключая текстовые и пустые значения), используйте следующую формулу массива.

```
=СУММ(ЕСЛИ(ЕЧИСЛО(Данные);1;0))
```

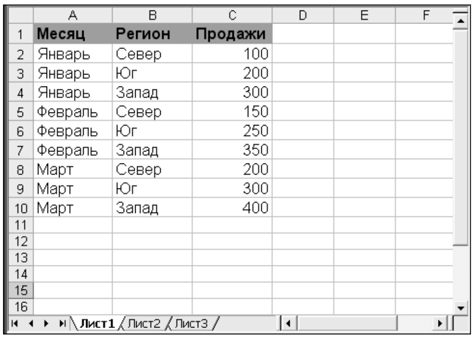
Для подсчета количества ячеек, содержащих ошибочные значения, применяйте такую формулу массива.

```
=СУММ(ЕСЛИ(ЕОШИБКА(Данные);1;0))
```

Чтобы подсчитать количество уникальных числовых значений (исключая текстовые; использовать пустые значения просто не разрешается), применяйте следующую формулу массива.

```
СУММ(ЕСЛИ(ЧАСТОТА(Данные;Данные)>0;1;0))
```

В табл. 3.4 представлены примеры формул массивов, которые находятся на рабочем листе, показанном на рис. 3.7.



	A	B	C	D	E	F
1	Месяц	Регион	Продажи			
2	Январь	Север	100			
3	Январь	Юг	200			
4	Январь	Запад	300			
5	Февраль	Север	150			
6	Февраль	Юг	250			
7	Февраль	Запад	350			
8	Март	Север	200			
9	Март	Юг	300			
10	Март	Запад	400			
11						
12						
13						
14						
15						
16						

Рис. 3.7. Эта простая база данных демонстрирует формулы массивов, используемые при подсчете и суммировании

На прилагаемом к книге компакт-диске содержится рабочая книга с примерами формул подсчета и суммирования.

Таблица 3.4. Сложные формулы массивов, использующие функцию СУММ

Формула массива	Возвращает
`=СУММ((A2:A10="Январь")*(B2:B10="Север")*C2:C10)`	Объем продаж за январь по северному региону
`=СУММ((A2:A10="Январь")*(B2:B10<>"Север")*C2:C10)`	Объем продаж за январь по всем регионам, кроме северного
`=СУММ((A2:A10="Январь")*(B2:B10="Север"))`	Количество продаж за январь по северному региону
`=СУММ((A2:A10="Январь")*(B2:B10="Север")+(B2:B10="Юг")))`	Количество продаж за январь в северном и южном регионах

Формула массива	Возвращает
`=СУММ((A2:A10="Январь")*(C2:C10>=200)*(C2:C10))`	Объем продаж за январь, каждая из которых составила не менее $200
`=СУММ((C2:C10>=300)*(C2:C10<=400)*(C2:C10))`	Объем продаж, каждая из которых составила не менее $300 и не более $400
`=СУММ((C2:C10>=300)*(C2:C10<=400))`	Количество продаж, каждая из которых составила не менее $300 и не более $400

Другие инструменты подсчета

Функция СЧЕТЕСЛИ используется при наличии только одного критерия подсчета. Если же условие является более сложным, то обратитесь к функции БСЧЕТ. Для этого необходимо представить информацию в виде базы данных (с именами полей в первой строке) и, кроме того, создать отдельный диапазон критериев, что позволяет указать их непосредственно на рабочем листе. Диапазон критериев также можно обрабатывать с помощью логических операторов ИЛИ, используя дополнительные строки. Подробно об этом рассказывается в справочной системе.

Если требуется подсчитать количество строк, отобранных с помощью инструмента Автофильтр, то воспользуйтесь функцией ПРОМЕЖУТОЧНЫЕ.ИТОГИ. Первый ее аргумент определяет тип промежуточного итога. Значение 3 представляет функцию СЧЕТЗ и возвращает количество видимых ячеек диапазона.

Если же подсчет необходимо вести на более серьезном уровне, то подумайте об использовании сводной таблицы (ознакомьтесь с этим средством, иначе вы не сможете воспользоваться одним из наиболее мощных инструментов Excel).

Работа со значениями даты и времени

Для хранения значений даты в Excel применяется система последовательной нумерации. Самой ранней датой, которую понимает программа Excel, является 1 января 1900 года. Этой дате соответствует число 1. Дата 2 января 1900 года равна следующему значению числовой последовательности — 2 и т.д.

Вам не придется анализировать, каким же числом представлена интересующая вас дата. Достаточно ввести дату в привычном формате, а Excel позаботится о ее корректной обработке. Например, если требуется задать дату 1 июня 1999 года, то просто введите **01.06.1999** (или используйте другой стандартный формат представления даты). Excel интерпретирует введенные данные и сохранит их в виде значения 36312, которое и является числовым значением для указанной даты.



В этой книге даты представлены в формате, характерном для русской версии программы Excel 2003. Он несколько отличается от принятого в США (и английской версии Excel) формата представления даты.

Ввод значений даты и времени

Работая со значениями времени, вы вводите в ячейку одно из них, пользуясь для этого одним из стандартных форматов. Система представления даты, применяемая в Excel, заключается в расширенном форматировании. При этом в него добавляется дробная часть, которая обозначает долю суток, отмеренную введенным временем. Другими словами, Excel представляет время, пользуясь для этого той же системой, что

и при представлении дат, независимо от того, в каких единицах измеряется это время: часах, минутах или секундах. Например, числовое значение даты 1 июня 1999 года составляет 36312. А вот полдень (середина суток) представлено значением 36312,5. Повторим, что вам не потребуется вводить дробные числовые значения для определенного времени суток.

Поскольку дата и время хранятся в виде числовых значений, то над ними допускается выполнять любые вычисления. Например, можно ввести формулу для подсчета количества дней между двумя датами.

```
=A2-A1
```

СОВЕТ

Когда дело доходит до подсчета времени, то ситуация усложняется. Если время вводится без даты, то в качестве даты берется 0 января 1900 года. Это не проблема, если только в результате подсчетов вы не получите отрицательное значение времени. В таком случае Excel выведет сообщение об ошибке (оно будет отображено как #########). Что же делать? Перейдите к формату дат 1904 году. Для этого выполните команду Сервис⇨Параметры, щелкните на вкладке Вычисления и установите флажок Система дат 1904. Не забывайте, что переход к этой системе может привести к неадекватному толкованию уже введенных в рабочем листе дат.

СОВЕТ

Складывая значения времени, вы обнаружите, что нельзя получить значение больше 24-х часов. Для каждого 24-часового периода программа Excel добавляет к дате еще один день. Решить данную проблему можно, изменив числовой формат: выделите квадратными скобками ту часть значения, в которой указано количество часов. Например, ниже представлен числовой формат, в котором может отображаться больше 24-х часов:

```
[чч]:мм
```

Использование дат до 1900 года

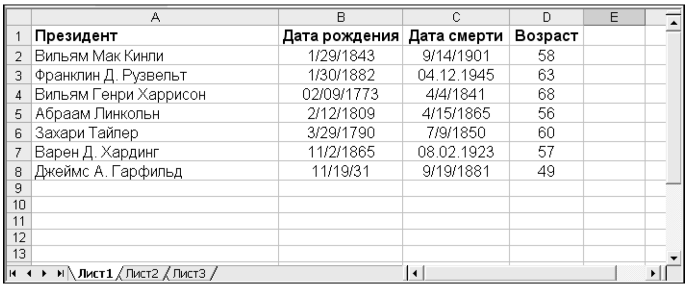
Как вы понимаете, мир начал свое существование не с 1 января 1900 года. И тем, кто использует Excel, работая с исторической информацией, часто приходится иметь дело с датами, намного более ранними, чем 1 января 1900 года. К сожалению, для управления такими датами подходит только один способ — их необходимо вводить в ячейку как текст. Например, Excel не будет против, если в ячейку вы введете такую дату.

```
4 июля 1776 года.
```

Впрочем, над датами, введенными как текст, нельзя проводить никаких операций. Например, изменить формат даты вы не сможете, как не сможете и определить день недели, на который эта дата приходится, а также подсчитать дату, отстоящую от текущей на семь дней.

НА КОМПАКТ-ДИСКЕ

На прилагаемом к книге компакт-диске содержится надстройка Extended Date Functions (Расширенные функции даты). Установив ее, вы получите доступ к восьми новым функциям рабочего листа, которые позволяют работать с любыми датами в диапазоне от 100-го до 9999-го года. На рис. 3.8 показан рабочий лист, в котором эти функции подсчитывают количество дней между датами до 1900 года.



Президент	Дата рождения	Дата смерти	Возраст
Вильям Мак Кинли	1/29/1843	9/14/1901	58
Франклин Д. Рузвельт	1/30/1882	04.12.1945	63
Вильям Генри Харрисон	02/09/1773	4/4/1841	68
Абраам Линкольн	2/12/1809	4/15/1865	56
Захари Тайлер	3/29/1790	7/9/1850	60
Варен Д. Хардинг	11/2/1865	08.02.1923	57
Джеймс А. Гарфильд	11/19/31	9/19/1881	49

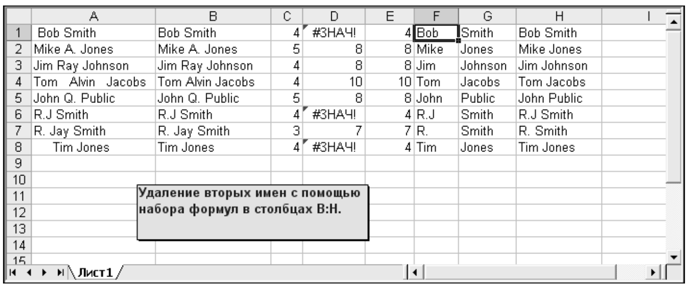
Рис. 3.8. Надстройка Extended Date Functions позволяет управлять датами до 1900 года

Создание мегаформул

Зачастую для получения необходимого результата в электронных таблицах используются промежуточные формулы, т.е. формула может зависеть от других формул, которые, в свою очередь, зависят еще от каких-либо формул. Добившись, чтобы все они работали правильно, вы получаете возможность удалить промежуточные формулы и применить вместо них единственную формулу, которая называется *мегаформулой*. Каковы ее преимущества? Используется меньше ячеек (и, следовательно, рабочий лист не так загроможден), а также быстрее происходит пересчет данных. Кроме того, понимающие пользователи будут поражены вашими “формулотворческими” способностями. Существуют ли в ней недостатки? Да, такую формулу совершенно невозможно понять или изменить.

Рассмотрим пример. Представьте себе рабочий лист со столбцом, в котором перечислены имена (и фамилии) людей. Предположим, что из этих имен с фамилиями следует убрать все вторые имена и инициалы. Дело только в том, что не у всех людей в списке есть такие имена и инициалы. Если редактировать ячейки вручную, то на это уйдут многие часы работы, поэтому вам просто необходимо прибегнуть к помощи формул. Данная задача не так уж и трудна, но для ее решения обычно требуется использовать несколько промежуточных формул.

На рис. 3.9 представлен результат довольно удачного решения — применено шесть промежуточных формул, перечисленных в табл. 3.5. Имена с фамилиями находятся в столбце `A`, а конечный результат — в столбце `H`. Что же касается столбцов от `B` до `G`, то в них как раз и содержатся промежуточные формулы.



A	B	C	D	E	F	G	H
Bob Smith	Bob Smith	4	#ЗНАЧ!	4	Bob	Smith	Bob Smith
Mike A. Jones	Mike A. Jones	5	8	8	Mike	Jones	Mike Jones
Jim Ray Johnson	Jim Ray Johnson	4	8	8	Jim	Johnson	Jim Johnson
Tom Alvin Jacobs	Tom Alvin Jacobs	4	10	10	Tom	Jacobs	Tom Jacobs
John Q. Public	John Q. Public	5	8	8	John	Public	John Public
R.J Smith	R.J Smith	4	#ЗНАЧ!	4	R.J	Smith	R.J Smith
R. Jay Smith	R. Jay Smith	3	7	7	R.	Smith	R. Smith
Tim Jones	Tim Jones	4	#ЗНАЧ!	4	Tim	Jones	Tim Jones

Рис. 3.9. Для удаления вторых имен и инициалов требуется шесть промежуточных формул

Таблица 3.5. Промежуточные формулы, введенные в первой строке рабочего листа Лист1 (рис. 3.9)

Ячейка	Промежуточная формула	Выполняемые функции
B1	`=СЖПРОБЕЛЫ(A1)`	Удаляет лишние пробелы
C1	`=НАЙТИ(" ";B1;1)`	Находит первый пробел
D1	`=НАЙТИ(" ";B1;C1+1)`	Находит второй пробел
E1	`=ЕСЛИ(ЕОШИБКА(D1);C1;D1)`	Использует первый найденный пробел, если не найден второй
F1	`=ЛЕВСИМВ(B1;C1)`	Выделяет первое имя
G1	`=ПРАВСИМВ(B1;ДЛСТР(B1)-E1)`	Выделяет последнее имя (т.е. фамилию)
H1	`=F1&G1`	Выполняет конкатенацию двух имен

Вы можете избавиться от всех промежуточных формул, если создадите мегаформулу. Для этого выполните следующее: создав промежуточные формулы, перейдите к конечной результирующей формуле и замените в ней каждую ссылку на ту или иную ячейку копией формулы, которая в этой ячейке находится (копия вводится без знака равенства). К счастью, для копирования и вставки можно использовать буфер обмена. Повторяйте эти действия до тех пор, пока в ячейке `H1` исчезнут все ссылки, кроме ссылок на ячейку `A1`. В конечном итоге у вас в одной ячейке получится следующая мегаформула.

```
=ЛЕВСИМВ(СЖПРОБЕЛЫ(A1);НАЙТИ(" ";СЖПРОБЕЛЫ(A1);1))
&ПРАВСИМВ(СЖПРОБЕЛЫ(A1);ДЛСТР(СЖПРОБЕЛЫ(A1))
-ЕСЛИ(ЕОШИБКА(НАЙТИ(" ";СЖПРОБЕЛЫ(A1);НАЙТИ(" ";
СЖПРОБЕЛЫ(A1);1)+1));НАЙТИ(" ";СЖПРОБЕЛЫ(A1);1);
НАЙТИ(" ";СЖПРОБЕЛЫ(A1);НАЙТИ(" ";СЖПРОБЕЛЫ(A1);1)+1)))
```

Когда вы убедитесь, что мегаформула работает, то можете удалить столбцы с промежуточными формулами, поскольку последние вам больше не понадобятся.

Мегаформула выполняет те же самые задачи, что и все промежуточные формулы, однако понять, как она работает, может только ее автор. Если вы собираетесь использовать мегаформулы, то перед их созданием проверьте, правильно ли работают промежуточные формулы. Или еще лучше — сохраните отдельно копию этих формул (на тот случай, если в расчетах обнаружится ошибка или в их алгоритм потребуется внести изменения).

НА ЗАМЕТКУ

Существует единственное ограничение, которое накладывается на мегаформулы: длина формулы Excel не может превышать 1024 символа. Решение этой проблемы заключается в создании на языке VBA пользовательской функции рабочего листа. Тогда мегаформулу можно заменить простой формулой, например.

```
NOMIDDLE(A1)
```

Я действительно написал такую функцию, чтобы сравнить ее с промежуточными формулами и мегаформулой.

Сложность мегаформул наталкивает на мысль, что при их использовании вычисления могут существенно замедляться. На самом деле это не так. Создадим рабочий лист, в котором мегаформула использовалась 65536 раз. Затем создадим другой рабочий лист, в котором применено шесть промежуточных формул. Полученные результаты сравним с работой VBA-функции. В табл. 3.6 приведены сравнительные результаты тестирования.

Таблица 3.6. Сравнение промежуточных формул с мегаформулой и VBA-функцией

Метод	Время пересчета (в секундах)	Размер файла
Промежуточные формулы	10,8	24,4 Мбайта
Мегаформула	6,2	8,9 Мбайта
VBA-функция	106,7	8,6 Мбайта

Результаты, получаемые на практике, будут существенно отличаться от указанных в таблице. Как именно — зависит от производительности системы и объема установленной памяти.

Итак, использование мегаформулы увеличивает скорость пересчета, а также *намного* уменьшает объем рабочей книги. VBA-функция оказалась во много раз медленнее — она осталась далеко позади остальных претендентов. Для VBA-функций это довольно типичная ситуация; они всегда выполняют операции медленнее, чем встроенные функции Excel.



На прилагаемом к книге компакт-диске можно найти все три файла, которые участвовали в этом тестировании.

Глава 4

Файлы Excel

В ЭТОЙ ГЛАВЕ...

В этой главе вы найдете описание следующих вопросов.

- Способы запуска Excel.
- Файлы, используемые и создаваемые Excel.
- Формат HTML.
- Использование в Excel системного реестра Windows.

Если вы собираетесь профессионально работать с Excel, то просто обязаны ознакомиться с несколькими способами запуска программы и понять, что же при этом происходит. Кроме того, вы должны иметь представление о типах файлов, используемых и генерируемых Excel. Именно этим вопросам посвящена данная глава.

Запуск Excel

Программу Excel можно запускать по-разному (в зависимости от способов ее установки). Но все варианты в конечном итоге сводятся к запуску исполняемого файла `Excel.exe`.

Запускаясь, программа Excel считывает из системного реестра Windows значения настроек и подключает все установленные надстройки. (Имеются в виду те надстройки, рядом с названиями которых в диалоговом окне Надстройки выставлены флажки.) Затем отображается пустая рабочая книга; количество ее листов определяется значением параметра, задаваемого пользователем и хранящегося в системном реестре. Это число можно изменить, выбрав на вкладке Общие диалогового окна Параметры опцию Листов в новой книге. Само диалоговое окно отображается в результате выполнения команды Сервис⇨Параметры.

Если в папке `Xlstart` (дословно означает “запуск Excel”) содержатся рабочие книги, то они автоматически откроются, а пустая книга не будет отображена на экране. Если же в этой папке находится файл рабочего пространства, то в специально подготовленном рабочем пространстве может открыться несколько файлов. Впрочем, вы всегда вправе определить другую папку для запуска. Для этого соответствующий путь указывается в поле Каталог автозагрузки вкладки Общие диалогового окна Параметры.

СОВЕТ

Предположим, что вам требуется изменить формат (или содержимое) пустой рабочей книги, которая открывается по умолчанию. Создайте стандартную рабочую книгу, измените ее должным образом и сохраните в папке `Xlstart` в виде шаблона с именем `Book.xlt`. Подробно о создании и использовании файлов шаблонов рассказывается в справочной системе программы.

Excel распознает несколько переключателей командной строки. Они описаны в табл. 4.1.

Таблица 4.1. Переключатели командной строки Excel

Переключатель	Операция
`/automation`	Excel запускается без подключения надстроек и шаблонов, а также загрузки файлов из папки `Xlstart` или любой другой папки автозагрузки. Этот переключатель используется для “чистой начальной загрузки” Excel
`/e`	Excel запускается во “упрощенном” режиме. Этот переключатель применяется тогда, когда требуется запустить программу Excel без вывода на экран надоедливой заставки и создания новой рабочей книги
`/embedded`	Запускает скрытый сеанс Excel (не рекомендуется)
`/m`	Программа создает новую рабочую книгу с единственным макролистом Excel 4.0 (устаревший объект)
`/o`	Excel регистрируется в системном реестре Windows. Заменяет пропущенные записи реестра. Неправильные записи реестра не исправляются (см. далее в этой таблице переключатель `/regserver`)
`/p` *папка*	Задает путь к папке, не указанной по умолчанию
`/r` *имя файла*	Программа открывает указанный файл в режиме только чтения
`/s`	Запускает Excel в “безопасном” режиме; не загружает надстроек и файлов, расположенных в папке `Xlstart` или другой папке автозапуска
`/regserver`	Excel регистрируется в системном реестре Windows, а затем заканчивает текущий сеанс работы. Используйте этот переключатель, если требуется, чтобы программа Excel перезаписала все параметры реестра и заново создала связи с необходимыми типами файлов, такими, например, как рабочие книги и диаграммы
`/unregserver`	Программа удаляет данные о себе из системного реестра Windows, а затем заканчивает сеанс работы

Указать один из этих переключателей можно, в частности, отредактировав свойства ярлыка, с помощью которого запускается программа Excel. Например, если требуется, чтобы папка `Xlfiles` была назначена папкой программы автозагрузки Excel по умолчанию, то с помощью переключателя `/p` вы выполните данную задачу в поле Рабочая папка, которое находится в диалоговом окне свойств для соответствующего ярлыка. Чтобы отобразить диалоговое окно Свойства, щелкните правой кнопкой мыши на ярлыке, а затем — левой кнопкой мыши на вкладке Ярлык. Например, содержимое поля Рабочая папка можно изменить следующим образом.

```
"C:\Program Files\Microsoft Office\Office\EXCEL.EXE" /p C:\Xlfiles
```

В Windows можно запускать несколько сеансов Excel. Каждый такой сеанс считается отдельным заданием. И, кроме того, многие пользователи добиваются немалых успехов, одновременно запуская разные версии программы Excel. Для этого устанавливайте версии программы строго в порядке выпуска их производителем.

Поддерживаемые форматы файлов электронных таблиц

Стандартным форматом файла Excel является рабочая книга XLS, однако данная программа открывает и сохраняет большое количество типов файлов, созданных другими приложениями.

Немаловажным является вопрос, может ли конкретный тип файла обеспечить сохранность данных. Другими словами, будет ли потеряна информация, если сохранить файл в определенном формате, а затем открыть этот файл в том же приложении? Как вы, возможно, и предполагаете, использование стандартного формата Excel (тип файлов XLS) гарантирует, что вы не потеряете абсолютно ничего — если, конечно, работаете с самой последней версией Excel.

Сохраняя и открывая файл, формат которого отличается от XLS (а также файл в старом формате XLS), вы рискуете потерять отдельные данные (обычно это форматирование и макросы, но иногда теряются формулы и диаграммы).

В последующих разделах рассказывается о типах файлов и о том, какие из них можно, а какие нельзя эффективно использовать в Excel.

Файлы электронных таблиц Lotus 1-2-3

Существует несколько разновидностей электронных таблиц Lotus.

- *Файлы WKS* — файлы одного рабочего листа, которые используются в Lotus 1-2-3 версии 1.*x* для DOS. В Excel их можно не только открывать, но и сохранять.

Кроме того, в Excel вы откроете файлы Microsoft Works, которые также имеют расширение WKS.

- *Файлы WK1* — файлы одного рабочего листа, которые используются в Lotus 1-2-3 версии 2.*x* для DOS. Форматирование этих файлов сохраняется в файлах с расширением `*.all` (сохраняются надстройкой Allways) или `*.fm1` (сохраняются надстройкой WYSIWYG (What You See Is What You Get, “что видишь, то и получаешь”). Программа Excel открывает и сохраняет все указанные файлы. Сохраняя файл в формате `*.wk1`, вы можете выбрать тип форматирования данных, а также расширение файла форматирования.
- *Файлы WK3* генерируются в Lotus 1-2-3 версии 3.*x* для DOS, Lotus 1-2-3 версии 4.*x* для DOS и Lotus 1-2-3 версии 1.*x* для Windows. В этих файлах может сохраняться больше одного листа. Форматирование хранится в файлах с расширением `*.fm3` (сохраняется надстройкой WYSIWYG). Программа Excel может читать и сохранять файлы WK3, к каждому из которых прилагается (не обязательно) свой файл FM3.
- *Файлы WK4* генерируются в Lotus 1-2-3 версии 4.*x* для Windows и Lotus 1-2-3 версии 5.*x* для Windows (компания Lotus, наконец-то, устранила главный недостаток — обязательность отдельного файла форматирования). В этих файлах может сохраняться больше одного листа. Программа Excel не открывает и не сохраняет такие файлы. Если вам необходимо открыть в Excel файл WK4, то воспользуйтесь единственным способом: с помощью Lotus 1-2-3 версии 4 для Windows (или более поздних) сохраните файл в формате WK3, который Excel понимает.
- *Файлы 123* генерируются Lotus 1-2-3 версии 97 и Lotus 1-2-3 Millennium Edition. В них сохраняется более одного листа. Excel также не открывает и не сохраняет эти файлы. Если вам потребуется прочитать в Excel файл формата 123, то для этого подойдет только один способ. Воспользовавшись Lotus 1-2-3, сохраните его в формате WK3, который программа Excel понимает.

Файлы электронных таблиц Quattro Pro

Файлы программы Quattro Pro также представлены несколькими версиями.

- *Файлы WQ1* — файлы одного рабочего листа, которые генерируются Quattro Pro для DOS версий 1, 2, 3 и 4. Эти файлы программа Excel может и открывать, и сохранять.
- *Файлы WQ2* генерируются Quattro Pro для DOS версии 5. Этот формат файлов программа Excel не может ни открывать, ни сохранять.
- *Файлы WB1* генерируются Quattro Pro для Windows версий 1 и 5 (версии со 2 по 4 отсутствуют). Этот формат программа Excel может только открывать, но не сохранять.
- *Файлы WB2* генерируются Quattro Pro для Windows версии 6. Этот формат файлов программа Excel не может ни открывать, ни сохранять.
- *Файлы WB3* генерируются Quattro Pro для Windows версий 7 и 8. Этот формат файлов программа Excel не может ни открывать, ни сохранять.



Загрузите с Web-узла компании Microsoft конвертор файлов Quattro Pro. Он позволит импортировать файлы WB3, созданные с помощью Quattro Pro 97.

Форматы файлов баз данных

Файлы DBF — это однотабличные файлы баз данных, генерируемые dBASE и некоторыми другими процессорами баз данных. Программа Excel открывает и сохраняет файлы DBF до формата dBASE 4 включительно.

Что касается других форматов баз данных, то открывать и сохранять их напрямую Excel не позволяет. Впрочем, для доступа к файлам других форматов баз данных можно использовать Microsoft Query, после этого скопируйте требуемые данные непосредственно на рабочий лист Excel или свяжите их с этим листом. Microsoft Query можно запустить непосредственно из Excel, выполнив команду Данные⇨Импорт внешних данных⇨Создать запрос.

Форматы текстовых файлов

В текстовых файлах данные сохраняются без форматирования. Для текстовых файлов представлено несколько стандартных форматов, но при этом не существует стандартных расширений файлов.

- В *файлах, разграниченных позициями табуляции,* каждая строка состоит из полей, которые отделены друг от друга символами табуляции. Программа Excel открывает эти файлы, преобразуя каждую строку текста в строку электронной таблицы, а каждое поле — в столбец. Кроме того, Excel сохраняет файлы в подобном формате, присваивая им по умолчанию расширение `*.txt`.
- В *файлах, разделяемых запятыми,* каждая строка состоит из полей, которые обычно отделены друг от друга запятыми. (Что же касается национальных настроек, в которых десятичные дроби содержат запятые, то тогда в текстовых файлах в качестве разделителей используются точки с запятой.) Иногда текст заключается в кавычки. Программа Excel открывает файлы, преобразуя каждую строку текста в строку электронной таблицы, а каждое поле — в столбец. Кроме того, Excel сохраняет эти файлы, присваивая им по умолчанию расширение CSV.

- В *файлах, разделяемых пробелами,* каждая строка состоит из полей, которые отделены друг от друга пробелами. Программа Excel открывает эти файлы, преобразуя каждую строку текста в строку электронной таблицы, а каждое поле — в столбец. Кроме того, Excel сохраняет эти файлы, присваивая им по умолчанию расширение PRN.

При открытии в Excel текстового файла, чтобы помочь вам определить его тип, в программе будет запущен мастер импорта текстовых данных.

СОВЕТ

Если вы не любите мастер импорта данных нужен, то отключите его: держите нажатой клавишу <Shift>, когда щелкаете на кнопке OK в диалоговом окне открытия документа.

НА ЗАМЕТКУ

Текстовые файлы также можно импортировать с помощью запросов. Для этого выполните команду Данные⇨Импорт внешних данных⇨Импортировать данные.

Другие форматы файлов

В Excel также поддерживаются следующие форматы файлов.

- *DIF* (*Data Interchange Format — формат обмена данными)* используется программой VisiCalc. Встречается достаточно редко. Такие файлы программа Excel может и открывать, и сохранять.
- *SYLK (SYmbolic LinK — символическая связь)* используется программой MultiPlan. В настоящее время файлы SYLK также являются большой редкостью. Программа Excel их может и открывать, и сохранять.

Файлы, сохраняемые в Excel

Excel сохраняет файлы нескольких типов. О них речь пойдет в этом разделе.

Файл XLS

Файлы рабочих книг XLS, создаваемые программой Excel 2003, поддерживают тот же формат, что Excel 2002, Excel 2000 и Excel 97. Эти файлы нельзя открыть ни в одной из версий Excel, предшествовавшей Excel 97. Впрочем, вы вправе сохранить рабочую книгу, указав любой из старых форматов Excel. При этом будет утеряна информация, специфичная для нового формата файла.

НА ЗАМЕТКУ

Рабочей книге или надстройке Excel можно присвоить любое расширение. Другими словами, эти файлы не обязательно должны иметь расширение XLS или XLA.

Какую версию имеет файл XLS

К сожалению, нет такого способа, который позволяет точно определить, в какой версии Excel создан тот или иной файл XLS. Если у вас установлена одна из ранних версий Excel, то при попытке открыть в ней файл XLS, созданный в более поздней версии, вы, скорее всего, получите сообщение об ошибке или увидите рабочую область, заполненную бессмысленным сочетанием символов. Но если файл все же успешно открыт, то с помощью простого оператора VBA можно легко определить версию Excel, в которой он создавался.

Откройте рабочую книгу и проверьте, что она осталась активной. Для запуска Visual Basic Editor (редактор Visual Basic) нажмите комбинацию клавиш <Alt+F11>, а затем, чтобы активизировать окно Immediate, — <Ctrl+G>. Перед тем, как нажать <Enter>, введите следующий оператор VBA:

```
Print ActiveWorkbook.FileFormat
```

В окне Immediate будет отображено значение, которое соответствует версии активной рабочей книги. Вы увидите одно из значений, перечисленных в следующей таблице.

Значение	Версия Excel
16	Excel 2
29	Excel 3
33	Excel 4
39	Excel 5, 95
-4143	Excel 97, 2000, 2002, 2003

Файл рабочего пространства

Файл рабочего пространства — это специальный файл, который содержит информацию о рабочем пространстве Excel. Например, в вашем проекте используются две рабочие книги, и окна этих книг должным образом настроены. Данную "оконную конфигурацию" можно сохранить в файле с расширением XLW. Для этого выполните команду Файл⇨Сохранить рабочую область. Впоследствии при открытии такого файла программа Excel восстановит требуемое рабочее пространство.

Важно понимать, что в файле рабочего пространства рабочие книги *не* сохраняются — хранятся только данные конфигурации, которые делают рабочие книги видимыми в окне Excel. Таким образом, если рабочее пространство следует передать в другой компьютер, то обязательно скопируйте не только файлы рабочих книг, но и файл XLW. Помните, что с помощью команды Файл⇨Сохранить рабочую область рабочие книги не сохраняются.

Файлы шаблонов

Каждую рабочую книгу можно сохранить в виде файла шаблона (с расширением XLT). Это свойство особенно пригодится, если вы регулярно создаете файлы, похожие друг на друга. Например, вам по долгу службы приходится готовить отчеты о ежемесячных продажах. Вы можете сэкономить время, создав шаблон с формулами и диаграммами, которые присутствуют в каждом отчете. На основе этого шаблона можно создавать новые файлы — для этого достаточно вставить только конечные значения.

Чтобы на основе имеющегося шаблона создать новую рабочую книгу, выполните команду Файл⇨Создать, а затем в диалоговом окне создания документа выберите необходимый шаблон.

В Excel 2002 и Excel 2003 такая последовательность команд приводит к отображению области задач Создание книги, поэтому вам придется выполнить дополнительные действия. Они заключаются в том, чтобы в области задач выбрать нужный файл шаблона, который также может загружаться с Web-узла компании Microsoft.

Если вы щелкнете на кнопке Создать, расположенной на панели инструментов, или нажмете комбинацию клавиш <Ctrl+N>, то выбрать шаблон не сможете. В результате будет создана рабочая книга с параметрами, установленными по умолчанию.

Если вы создали шаблон `Book.xlt`, то на его основе будут созданы новые рабочие книги по умолчанию. Кроме того, можно создать шаблон `Sheet.xlt`, на основе которого будут создаваться рабочие листы, добавляемые в рабочую книгу. Обратите внимание, что шаблон для листов диаграмм создать нельзя, поскольку программа Excel управляет шаблонами диаграмм несколько по-другому, чем шаблонами рабочих листов.

В локальном компьютере шаблоны могут сохраняться в нескольких местах.

- В *папке Xlstart*. Именно здесь хранятся автоматически подключаемые шаблоны `Book.xlt` и `Sheet.xlt`. Кроме того, в эту папку можно помещать другие шаблоны рабочих книг.
- В *папке Templates*. Здесь хранятся те шаблоны рабочих книг, названия которых отображаются в диалоговом окне создания документа.

СОВЕТ

Местоположение папки Templates зависит от используемой версии Excel. Чтобы найти его, выполните следующий оператор VBA:

```
MsgBox Application.TemplatesPath
```

Файлы панелей инструментов

Конфигурации панелей инструментов и строки меню хранятся программой Excel в файле с расширением XLB. Когда пользователь завершает работу в Excel 2003, текущая конфигурация сохраняется в файле `Excel10.xlb`. Точное расположение и имя этого файла зависят от используемой версии Excel; сам файл вы найдете, выполнив на жестком диске поиск по расширению `*.xlb`. В этом файле находятся данные о расположении и состоянии всех панелей инструментов (включая пользовательские) и строк*и* меню, а также данные об изменении встроенных панелей инструментов и меню.

Перечень имен и путей расположения файлов XLB, используемых различными версиями Excel, можно найти в справочной системе программы Excel 2003. Эту информацию ищите по ключевому слову `xlb`.

СОВЕТ

Если файл XLB повредится, то Excel не сможет загрузиться. Если возникает ситуация сбоя программы при загрузке, то переименуйте файл XLB. Программа автоматически создаст новый файл с настройками по умолчанию. Все дополнительные настройки будут при этом потеряны — вам придется заново перенастраивать интерфейс программы.

Файлы надстроек

Надстройка — это файл рабочей книги, имеющей несколько важных особенностей.

- Если в надстройке свойство `IsAddin` имеет значение `True` (Истина), то воспользуйтесь командой Сервис⇨Надстройки.
- Надстройка считается скрытой рабочей книгой, и отобразить ее в окне Excel пользователь не сможет. Следовательно, надстройка никогда не бывает активной рабочей книгой.
- Рабочая книга надстройки не входит в коллекцию `Workbooks` (Рабочие книги).

Многие надстройки дополняют программу Excel новыми возможностями и функциями. Эти новые возможности довольно легко использовать; их порой невозможно отличить от встроенных средств программы.

На основе файлов рабочих книг вы можете создать собственные надстройки. Следует отметить, что отдельные приложения Excel, распространяемые среди пользовате-

лей, предпочтительно создавать в виде надстроек. По умолчанию для них использует-ся расширение XLA, но вы можете применить и другое расширение.

В Excel поддерживаются не только надстройки в формате XLA, но также XLL и (начиная с Excel 2000) COM. Надстройки двух последних типов создаются не в Excel. Что же касается нашей книги, то в ней обсуждаются лишь надстройки XLA.

Подробно о надстройках рассказывается в главе 21.

Excel и HTML

HTML — это язык World Wide Web. При просмотре документов, помните, что загружаемые вашим браузером данные обычно имеют формат HTML. HTML-файл состоит из текстовой информации и специальных дескрипторов, задающих форматирование текста. Браузер интерпретирует дескрипторы, выполняет по ним форматирование данных и отображает конечный результат на экране.

Если говорить об Excel 2000 и более поздних версиях, то в них HTML можно использовать в качестве "родного" формата файлов. Другими словами, рабочую книгу можно сохранить в формате HTML, а затем открыть полученный HTML-файл в Excel, и он будет выглядеть точно так же, как перед сохранением. Вся информация, специфичная для данных Excel (например, макросы, диаграммы, сводные таблицы и параметры рабочих листов), только теоретически остается неизменной. HTML является относительно простым форматом файлов. Правда, утверждение, что рабочая книга Excel может выдержать подобное "преобразование" без потерь данных, является лишь теоретическим обоснованием, но никак не доказанным фактом.

Конечно, на использование HTML в качестве "родного" формата файлов могут решиться только весьма доверчивые пользователи, поскольку разработчики Microsoft, на наш взгляд, больше разрекламировали предоставленную возможность, чем "довели ее до ума". В реальных проектах, если не считать редких исключений, пользы от применения HTML будет немного.

Так как же это работает?

Лучший способ понять, каким образом Excel может использовать HTML в качестве "родного" формата файлов, заключается в проведении небольшого творческого эксперимента. Начните с создания рабочей книги, содержащей только один рабочий лист. Введите в него несколько значений и формул, выполните простое форматирование, а затем сохраните рабочую книгу в формате HTML. Для этого используйте команду Файл⇨Сохранить как веб-страницу, при этом обязательно выставьте переключатель всю книгу. На рис. 4.1 показана простая рабочая книга, в которую введено два значения и одна формула, причем в ячейке с формулой задано форматирование полужирным начертанием. Эта рабочая книга поможет нам в изучении HTML-файлов, сохраняемых в Excel.

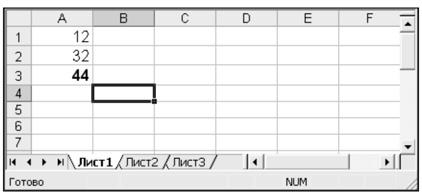


Рис. 4.1. Сохраните простую рабочую книгу в формате HTML

Материал оставшейся части этого раздела создавался с расчетом того, что вы уже знакомы с HTML.

Откройте в браузере по умолчанию полученный HTML-файл. Этот файл будет подобен исходной рабочей книге. Впрочем, открытый вами HTML-файл является “статическим” документом, лишенным интерактивности. Для просмотра его HTML-кода используйте команду Вид⇨Просмотр HTML-кода. Вы, возможно, будете весьма удивлены. Даже профессионалы в области разработки HTML-документов считают, что этот “простой” Web-документ весьма сложен.

Ниже приведены некоторые особенности HTML-файла, создаваемого в Excel.

- Всю рабочую книгу Excel можно представить в виде единственного HTML-файла. Другими словами, в HTML-файле находится вся необходимая информация для создания точной копии первоначальной рабочей книги. Впрочем, такое бывает не всегда. Далее вы узнаете, когда простого HTML-файла недостаточно.
- Бóльшая часть документа находится между дескрипторами `<head>` и `</head>`.
- Значительную часть составляют определения стилей. Это информация, которая находится между дескрипторами `<style>` и `</style>`, расположенными, в свою очередь, между `<head>` и `</head>`.
- Что касается “реального” текста, который отображается в браузере, то он располагается в таблице (между дескрипторами `<table>` и `</table>`).
- Формула сохраняется с помощью специального аргумента дескриптора `<td>`. Браузеры игнорируют этот аргумент, но его информация будет использоваться программой Excel при очередном открытии файла.

Размер HTML-файла, созданного для этой простой рабочей книги, превышает 4000 байтов, что очень много для Web-страницы с небольшим количеством полезной информации, которая отображается в браузере. Информация, которая непомерно увеличивает размер файла, используется Excel для создания рабочей книги при очередном открытии HTML-файла.

Усложнение HTML-документа

Та рабочая книга, которая использовалась в предыдущем разделе, является предельно простой. Теперь добавим в нее сложные объекты и посмотрим, что же произойдет с HTML-файлом.

Используя файл созданной ранее простой рабочей книги, выберите диапазон `A1:A3` и нажмите <F11>, чтобы создать новый лист диаграммы. Снова сохраните файл и загрузите его в своем браузере. Вы увидите, что он подобен рабочей книге Excel (это относится даже к расположенным внизу вкладкам листов и навигационным кнопкам!).

Теперь размер HTML-файла увеличился более чем вдвое (и уже составляет примерно 10 000 байтов). Важнее то, что в папке с сохраненным файлом появилась вложенная папка с дополнительными файлами (их шесть, если говорить о рассматриваемой простой рабочей книге). Файлы, которые находятся в этой подпапке, необходимы для отображения в браузере копии рабочей книги и для повторного создания рабочей книги при очередном открытии HTML-файла в Excel.

Проверив HTML-файл, вы увидите, что он стал значительно сложнее, чем был вначале. В нем появился сложный JavaScript-код (JavaScript — язык написания сценариев, поддерживаемый браузерами Internet Explorer и Netscape Navigator). Теперь

HTML-файл значительно сложнее, особенно для понимания рядового разработчика HTML-кода. И это все, не учитывая другие файлы, сохраненные во вложенной папке.

- Три HTML-файла (по одному на каждый из листов, а также файл, который отображает панель со вкладками).
- GIF-файл (диаграмма).
- CSS-файл (каскадная таблица стилей, содержащая информацию о форматировании данных).
- XML-файл. XML — это сокращение от eXtensible Markup Language, т.е. расширяемый язык разметки документов (он не рассматривается в этой книге). Итак, ситуация усложняется в геометрической прогрессии.

Возможно, вам потребуется открыть другие рабочие книги Excel и сохранить их в виде HTML-файлов. Вскоре вы обнаружите, что в подпапке создается файл еще одного типа — MSO (MSO в данном случае расшифровывается как "файл Microsoft Office"). Это битовый файл с информацией, которая используется для воссоздания специфичных для Excel объектов (таких, например, как макросы, сводные таблицы, условное форматирование и т.д.).

Как вы уже, возможно, догадались, сохранение рабочей книги Excel в формате HTML вызывает немало проблем. Например, если файл необходимо перенести в другое место, то вместе с ним обязательно следует переносить и все дополнительные файлы. Если один из дополнительных файлов поврежден, Excel не сможет воссоздать рабочую книгу. Открытие и сохранение HTML-файлов происходит намного медленнее, чем открытие и сохранение обычных XLS-файлов. Отсюда вывод: сохраняйте свои рабочие книги в формате HTML лишь тогда, когда имеете на то веские причины.

А как насчет интерактивности?

Если HTML вам еще не надоел, то настало время перейти на следующий уровень сложности. В Excel можно сохранять HTML-файлы, которые представляют электронные таблицы, содержащие интерактивность. Другими словами, когда HTML-файл отображается в браузере, пользователь может взаимодействовать с документом как с электронной таблицей — вводить данные, изменять формулы, настраивать форматирование ячеек, просматривать "живые" диаграммы и даже перетаскивать данные в сводных таблицах. Эта возможность, которая называется *публикацией* (в отличие от обычного *сохранения*), ограничена тем, что обеспечивается лишь при сохранении одного листа (а не всей рабочей книги).

В Excel 2003 вы можете создать интерактивный HTML-файл на основе многолистной рабочей книги. В предыдущих версиях программы сохранять в формате HTML можно было только один интерактивный рабочий лист.

Чтобы понять, как в HTML обеспечивается интерактивность, активизируйте лист, содержащий формулы. Выполните последовательность команд Файл⇨Сохранить как веб-страницу. В диалоговом окне сохранения документа выставьте переключатель Выделенное: Лист, а также установите флажок Добавить интерактивность. Щелкните на кнопке Опубликовать. Появится диалоговое окно Публикация веб-страницы. Ничего не изменяя в этом окне, опять щелкните на кнопке Опубликовать.

Интерактивные HTML-документы, созданные в Excel, открываются только в Internet Explorer.

Открыв в браузере по умолчанию HTML-файл, вы увидите, что на странице отображается объект, который похож на электронную таблицу и который действительно является интерактивным. На рис. 4.2 представлен пример, полученный с помощью браузера Microsoft Internet Explorer.

А как же Script Editor?

К сожалению, в этой книге не уделено внимание достаточно важной теме, имеющей непосредственное отношение к Excel, — не рассмотрено средство Microsoft Script Editor (редактор сценариев Microsoft). Доступ к этому инструменту можно получить, нажав комбинацию клавиш <Alt+Shift+F11>. Редактор сценариев используется для редактирования в HTML-документе кода JavaScript (или VBScript). В настоящей книге, на наш взгляд, эту тему рассматривать нецелесообразно, т.к. она может вызвать интерес достаточно ограниченного количества пользователей. Поэтому основное внимание будет уделено подлинной основе Excel — языку VBA и тем приложениям, которые не предназначены для публикации в Web.

Вы, скорее всего, ожидаете, что HTML-файл, сгенерированный в виде интерактивного рабочего листа, должен быть намного сложнее, чем пример из предыдущего раздела. Вот тут вы не правы. Интерактивный рабочий лист занимает всего лишь один HTML-файл. По причине того, что для публикации взят только один лист, создавать на странице панель вкладок не потребуется. Бремя интерактивности возложено на элементы управления ActiveX. Поэтому, чтобы конечный пользователь имел возможность просматривать интерактивный Excel-файл, в его компьютере должен быть установлен пакет Office 2000 или более поздняя его версия.



Задача данного раздела — привести краткий обзор возможностей формата HTML, поддерживаемых в Excel 2000-2003. Эта тема достойна отдельной книги, писать которую я не имею никакого желания (в том числе и благодаря письмам читателей).

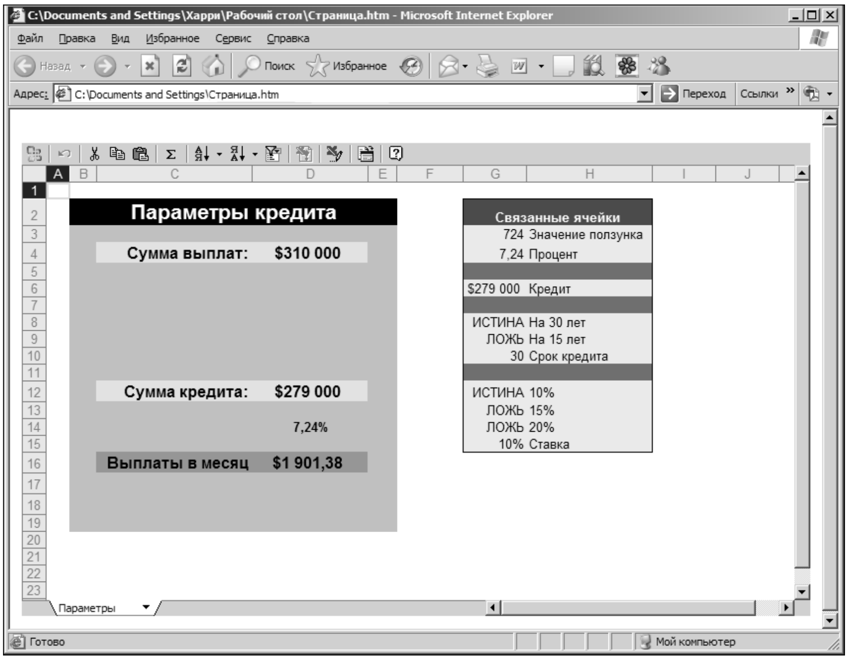


Рис. 4.2. Пример интерактивного рабочего листа Excel, отображаемого в браузере

Импорт и экспорт XML-данных

К важным особенностям Excel 2003 относится возможность импортирования и экспортирования XML-данных. В этом разделе мы вкратце остановимся на этих двух операциях.

НОВИНКА

Этот раздел в полной мере смогут изучить только пользователи профессиональной версии Excel 2003. Отдельные XML-файлы можно открыть с помощью команды Файл⇨Открыть. В предыдущих версиях программы эта операция не выполнялась.

Что такое XML

XML — это общепризнанный стандарт обмена данными между приложениями. XML — это язык разметки документов, подобный HTML, но обладающий более широкими возможностями.

В XML используются дескрипторы, с помощью которых определяются все элементы документа. С помощью XML-дескрипторов задается структура объектов, а также их назначение. В отличие от HTML-дескрипторов, которые определяют только форматирование (вид) документа, XML позволяет определить содержимое и структуру файла. Стоит заметить, что в XML четко разграничиваются понятия содержимого и форматирования документа.

Ниже приведен пример XML-файла, который содержит данные почтового сообщения.

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
<to>Bill Smith</to>
<from>Mark Jackson</from>
<subject>Meeting date</subject>
<body>The meeting will be at 8:00 a.m. on Tuesday</body>
</message>
```

При просмотре файла в Internet Explorer четко просматривается структура документа (рис. 4.3).

НА КОМПАКТ-ДИСКЕ

Все файлы этого примера приведены на прилагаемом к книге компакт-диске.

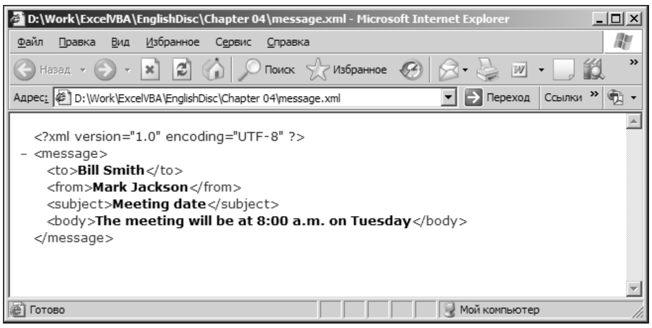


Рис. 4.3. XML-файлы отображаются с помощью браузера Internet Explorer

В отличие от HTML, спецификация XML не имеет строго определенного набора дескрипторов. Наоборот, с помощью XML создаются новые дескрипторы и образуются связи между ними. Поскольку в XML нет строго заданных дескрипторов, то вы можете моделировать практически любой тип документов.



Материал этого раздела не предусматривает описание основ работы с XML. В нем представлена только краткая характеристика возможностей спецификации. Вы найдете огромное количество книг по XML, которые помогут вам разобраться с этой технологией.

В следующих трех подразделах мы рассмотрим простейшие примеры использования XML-данных в Excel.

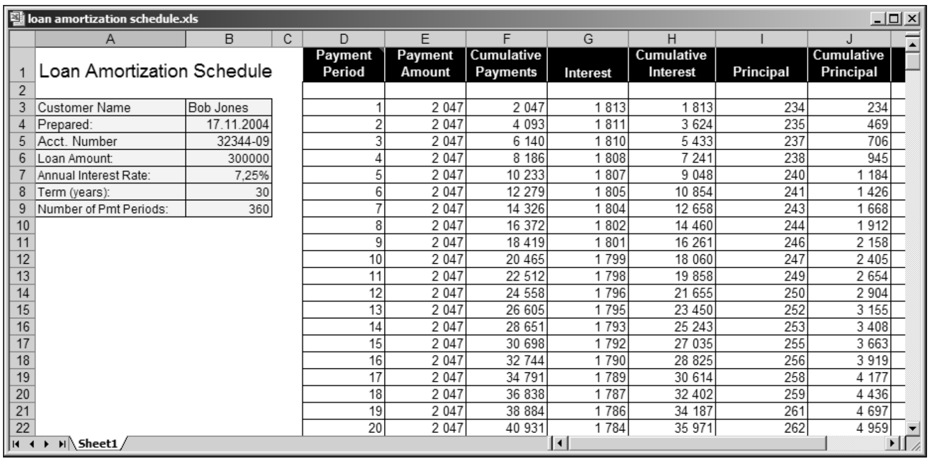
Импортирование XML-данных

Давайте рассмотрим пример, показанный на рис. 4.4. В этом рабочем листе в столбце B отображаются параметры погашения займа. Предположим, что серверная компьютерная система генерирует XML-файлы, каждый из который содержит сведения об отдельном клиенте. Ниже приведен пример такого файла.

```
<?xml version="1.0"?>
<Customer>
<Name>Joe Smith</Name>
<AccNo>32374-94</Name>
<LoanAmt>$325,983</LoanAmt>
<IntRate>6,25%</IntRate>
<Term>30</Term>
</Customer>
```

В этом файле определено пять элементов: `Name`, `AccNo`, `LoanAmt`, `IntRate` и `Term`. Два других поля (`Prepared` и `Number of Pmt Periods`) вычисляются с помощью формул, а потому не рассматриваются как элементы данных.

Первым делом вам нужно определить карту XML. Для начала убедитесь, что на экране отображается область задач Источник XML. (Выберите команду Данные⇨XML⇨Источник XML.)


loan amortization schedule.xls

	A	B	C	D	E	F	G	H	I	J
1	Loan Amortization Schedule			Payment Period	Payment Amount	Cumulative Payments	Interest	Cumulative Interest	Principal	Cumulative Principal
2										
3	Customer Name	Bob Jones		1	2 047	2 047	1 813	1 813	234	234
4	Prepared:	17.11.2004		2	2 047	4 093	1 811	3 624	235	469
5	Acct. Number	32344-09		3	2 047	6 140	1 810	5 433	237	706
6	Loan Amount:	300000		4	2 047	8 186	1 808	7 241	238	945
7	Annual Interest Rate:	7,25%		5	2 047	10 233	1 807	9 048	240	1 184
8	Term (years):	30		6	2 047	12 279	1 805	10 854	241	1 426
9	Number of Pmt Periods:	360		7	2 047	14 326	1 804	12 658	243	1 668
10				8	2 047	16 372	1 802	14 460	244	1 912
11				9	2 047	18 419	1 801	16 261	246	2 158
12				10	2 047	20 465	1 799	18 060	247	2 405
13				11	2 047	22 512	1 798	19 858	249	2 654
14				12	2 047	24 558	1 796	21 655	250	2 904
15				13	2 047	26 605	1 795	23 450	252	3 155
16				14	2 047	28 651	1 793	25 243	253	3 408
17				15	2 047	30 698	1 792	27 035	255	3 663
18				16	2 047	32 744	1 790	28 825	256	3 919
19				17	2 047	34 791	1 789	30 614	258	4 177
20				18	2 047	36 838	1 787	32 402	259	4 436
21				19	2 047	38 884	1 786	34 187	261	4 697
22				20	2 047	40 931	1 784	35 971	262	4 959

Sheet1

Рис. 4.4. В рабочем листе используются импортируемые XML-данные

Для добавления карты XML выполните следующие действия.

1. Щелкните на кнопке Карты XML в нижней части области задач. На экране появится диалоговое окно Карты XML.
2. Щелкните на кнопке Добавить для отображения диалогового окна Выберите источник XML.
3. Укажите один из XML-файлов. Какой именно — не имеет значения. Он будет использоваться для построения схемы.
4. Щелкните на кнопке OK. В области задач появятся элементы XML-файла (рис. 4.5).

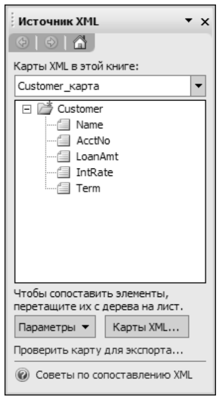


Рис. 4.5. Область задач отображает список элементов XML-файлов

Далее вставьте элементы данных в соответствующие ячейки рабочего листа.

1. В области задач щелкните на элементе `Name` и перетащите его в ячейку `B3`.
2. Переместите элемент `AccNo` в ячейку `B5`.
3. Переместите элемент `LoanAmt` в ячейку `B6`.
4. Переместите элемент `IntRate` в ячейку `B7`.
5. Переместите элемент `Term` в ячейку `B8`.

Наконец, вы можете приступать к импортированию XML-файла. Выполните команду Данные⇨XML⇨Импорт. Выберите XML-файл. Данные из XML-файла будут подставлены в соответствующие ячейки рабочего листа. Для расчета другого плана амортизации импортируйте другой XML-файл.

Импортирование XML-данных в список

Предыдущий пример описывает импорт XML-данных, состоящих всего их одной записи. Далеко не всегда XML-файлы состоят только из одной записи. Если в файле несколько записей данных, то они называются *повторяющимися элементами*. Примером такого файла может быть список кредитных сведений для нескольких клиентов.

Для импортирования XML-файла с повторяющимися элементами воспользуйтесь командой Файл⇨Открыть. Укажите файл в диалоговом окне Открытие документа и щелкните на кнопке Открыть. Появится диалоговое окно Открытие XML, показанное на рис. 4.6. В нем приведено три переключателя.

- XML-список. Excel преобразует данные в список.
- Книга, доступная только для чтения. Данные импортируются в рабочую книгу, но последняя приобретает атрибут "только чтение".
- Использовать область задач XML-источника. Excel использует схему XML-файла, не импортируя его данные. Для импортирования элементов применяется область задач.

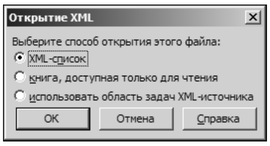


Рис. 4.6. Диалоговое окно Открытие XML

На рис. 4.7 показаны импортированные в рабочий лист XML-данные.

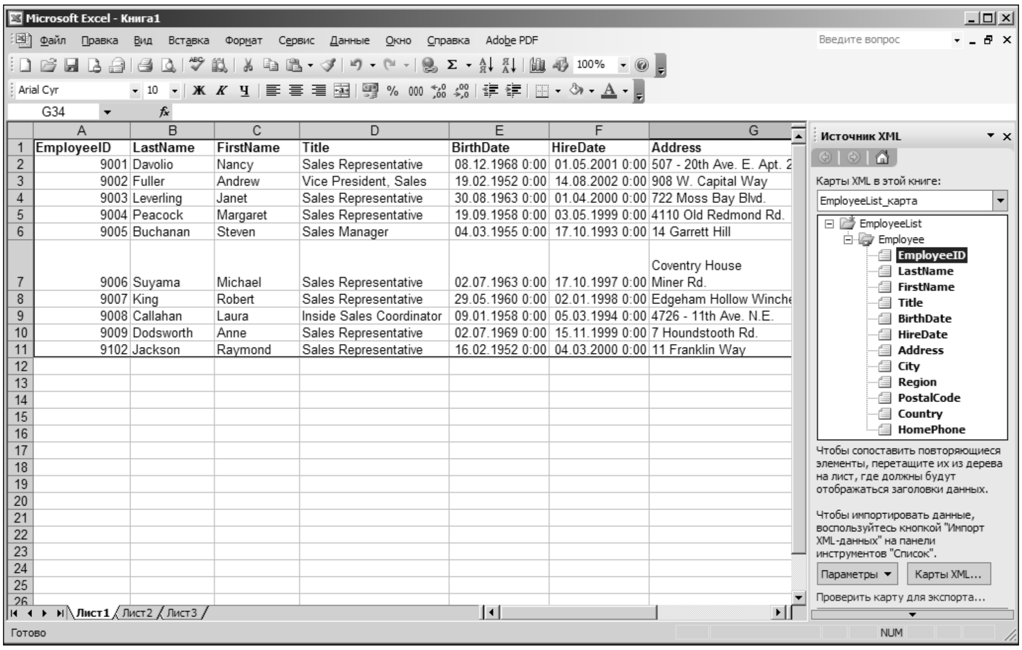


Рис. 4.7. Импортированные XML-данные

НА ЗАМЕТКУ

При отображении в области задач повторяющиеся элементы имеют другие значки.

Экспортирование XML-данных в Excel

Для того чтобы экспортировать данные в XML-файл, вам необходимо добавить в рабочую книгу карту XML. И эта карта должна соответствовать вашим данным. Далее выполните команду Данные⇨XML⇨Экспорт.

К сожалению, невозможно экспортировать в формат XML произвольный диапазон данных. Например, если вы создаете в рабочей книге список, то не сможете экспортировать в XML-файл только его (за исключением добавления в файл соответствующей карты XML). Невозможно изменить карту после создания XML-файла с помощью Excel.

ПЕРЕКРЕСТНАЯ ССЫЛКА

В главе 27 описана простая процедура создания XML-файла на основе диапазона данных.

НА ЗАМЕТКУ

Если воспользоваться командой Файл⇨Сохранить как, то можно выбрать для сохраняемого файла формат Таблица XML. В результате будет создан файл, использующий карту Microsoft XMLSS. Этот файл не совместим с другими приложениями, поддерживающими формат XML.

Параметры Excel в системном реестре

В этом разделе изложены основные сведения о системном реестре Windows. Кроме того, вы узнаете, каким образом программа Excel использует реестр для хранения своих настроек.

О системном реестре

В Windows 3.1 для хранения сведений о связывании типов файлов с определенными приложениями и OLE-регистрации использовалась база данных регистрации. Что же касается Windows 95 (и более поздних версий), то в ней концепция регистрации была расширена до уровня системного реестра, хранящего данные конфигурации всех приложений, а также настройки самого компьютера.

Системный реестр — это иерархическая база данных, доступная для прикладных программ. Информация этой базы хранится в двух файлах: `System.dat` (для системных данных) и `User.dat` (для настраиваемых пользователем данных). Оба файла находятся в папке `Windows`. Кроме того, системным реестром может использоваться файл `Policy.pol` — файл, содержащий системные политики, данные которых заменяют информацию файлов реестра.

Для просмотра системного реестра воспользуйтесь редактором реестра (это файл `Regedit.exe`, который находится в папке Windows). Впрочем, к этой программе вы можете обратиться также для редактирования содержимого системного реестра. Подумайте, прежде чем начать изменение реестра, отдав себе полный отчет в выполняемых действиях. Вначале прочтите врезку "Перед тем, как редактировать системный реестр...". На рис. 4.8 показано окно редактора реестра.

Как уже отмечалось, системный реестр является иерархической структурой. Он состоит из разделов и параметров.

```
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
```

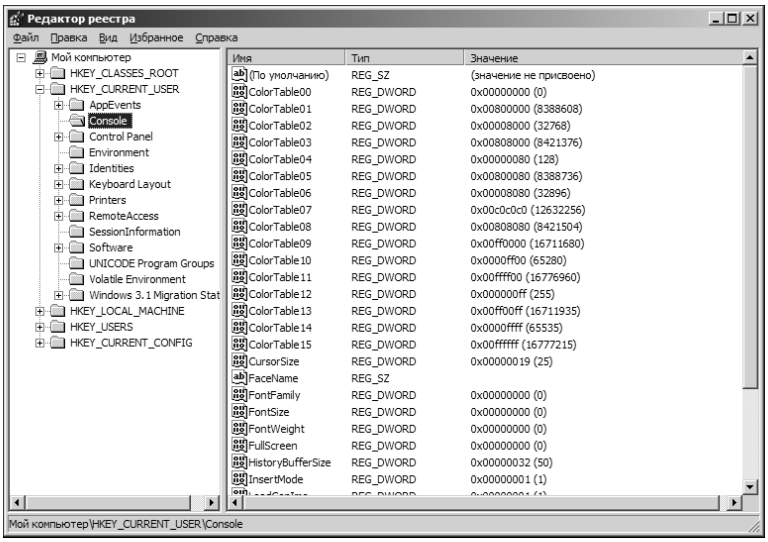


Рис. 4.8. Просматривать системный реестр и изменять в нем данные можно с помощью специального редактора

Параметры Excel

Информация, используемая программой Excel 2003, в системном реестре хранится в разделе

```
HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Excel
```

В этом разделе системного реестра находятся параметры, которые содержат конкретные значения, предопределяющие поведение Excel в любых ситуациях.

Перед тем, как редактировать системный реестр…

С помощью программы `Regedit.exe` можно изменять данные в системном реестре, в том числе информацию, которая представляет особую важность для работы вашей системы. Другими словами, если вы сделаете изменения не там, где это требуется, то работоспособность системы будет нарушена.

Поэтому всегда соблюдайте простые меры предосторожности. Во-первых, убедитесь, что у вас есть дискета аварийной загрузки системы. (Эту дискету можно создать с помощью аплета Установка и удаление программ, запускаемого из папки Панель управления.) Дискета аварийной загрузки используется также для запуска Windows в случаях повреждения системы.

Во-вторых, обязательно выполняйте в редакторе реестра команду Файл⇨Экспорт. С ее помощью вы сможете сохранить ASCII-версию всего реестра или отдельной его ветви. И если вы чувствуете, что запутались во многочисленных параметрах, то всегда имеете возможность, импортировав необходимый ASCII-файл, вернуть системный реестр в предыдущее состояние (с помощью команд Файл⇨Импорт). Более подробно об этом рассказано в справочной системе программы Regedit.

Параметры системного реестра, регулирующие поведение Excel, автоматически обновляются при ее закрытии.

Важно понять, что программа Excel считывает значения системного реестра только один раз — при запуске. Кроме того, Excel обновляет значения в реестре лишь в процессе нормального завершения работы. Если по вине Excel в системе происходит сбой (к сожалению, это не такой уж редкий случай), то информация в системном реестре не обновляется. Например, если вы измените значение одного из параметров Excel (пусть этим параметром будет отображаемость строки состояния), то новое значение в системный реестр не будет записано, пока вы не закроете окно Excel стандартным способом.

В табл. 4.2 перечислены разделы системного реестра, которые регулируют работу Excel 2003. Вполне возможно, что не все из них содержатся в базе данных вашего реестра.

Таблица 4.2. Информация в системном реестре, относящаяся к Excel

Раздел	Описание
Add-in Manager (Диспетчер надстроек)	Содержит перечень надстроек, который выводится на экран, при выполнении команды Сервис⇨Надстройки. В этом перечне отсутствуют надстройки, которые поставляются вместе с Excel. Если в указанном списке приведена надстройка, которую вы уже не используете, то соответствующий элемент списка можно удалить с помощью редактора реестра

Раздел	Описание
Converters (Конверторы)	Содержит перечень дополнительных (внешних) конверторов файлов, которые не встроены в Excel
AutoSave (Автосохранение)	Содержит установленное вами значение параметра автосохранения
Delete Commands (Удаление команд)	Позволяет указать, какие команды меню не должны отображаться на экране
Error Checking (Проверка ошибок)	Содержит значения тех параметров, которые относятся к проверке формул на наличие ошибок
Init Commands (Команды начальной загрузки)	Содержит информацию о пользовательских командах
Init Menus (Меню начальной загрузки)	Содержит информацию о пользовательских меню
Line Print (Печать строк)	Содержит значения тех параметров, которые применяются при использовании макросов Lotus 1-2-3. Excel обновляет этот раздел каждый раз, когда выполняет макрос Lotus 1-2-3, имеющий параметр `/wgdu` (Worksheet Global Default Update, “общее обновление рабочих листов, выполняемое по умолчанию”)
Options (Параметры)	Раздел, состоящий из большого количества настроек. В нем представлены значения самых разных параметров, в том числе пути к файлам, автоматически открывающимся при запуске Excel (например, к файлам надстроек)
Recent Files (Недавно сохраненные файлы)	Сохраняет имена последних сохраненных файлов (до девяти элементов)
Resiliency (Восстанавливаемость)	Информация, используемая для восстановления документов
Security (Безопасность)	Указывает уровень безопасности для открытия файлов, содержащих макросы
Spell Checker (Проверка орфографии)	Хранит выбранные вами значения для настройки системы проверки орфографии
UserInfo (Информация о пользователе)	Информация о пользователе

Несмотря на то, что значения большинства параметров можно изменить в диалоговом окне Параметры программы Excel, некоторые специальные (но не менее полезные) настройки редактируются исключительно в системном реестре (в Excel они недоступны).



Вновь напомним: прежде чем приступить к изменениям в системном реестре, обратите внимание на врезку “Перед тем, как редактировать системный реестр…”.

Часть II

Разработка приложений Excel

В этой части...

Глава 5

Приложения электронных таблиц

В ЭТОЙ ГЛАВЕ...

В этой главе речь пойдет об особенностях использования электронных таблиц в реальной жизни. Данный вопрос является одним из основных для всей книги. В самом начале важно определить, сколько же усилий потребуется для разработки конкретного проекта. Когда вы закончите читать эту главу, будете хорошо понимать, что я имел в виду, говоря о приложении электронных таблиц. После изучения оставшихся глав книги самостоятельно создайте такое приложение — в дальнейшем вы не будете иметь проблем с разработкой в Excel собственных приложений электронных таблиц. Но вначале обратимся к основам.

- Рабочее определение приложения электронных таблиц
- Разница между пользователем электронных таблиц и их разработчиком
- Классификация пользователей электронных таблиц, которая поможет определить аудиторию вашего приложения
- Обсуждение причин, по которым пользователи обращаются к электронным таблицам
- Классификация основных типов электронных таблиц

Наверняка уже несколько лет вы работаете с электронными таблицами, однако, скорее всего, ваша основная задача — это создание обычной электронной таблицы. И все. Вы, вероятно, не задумывались о более глобальных вопросах, изложенных далее в этой главе. Речь идет о разных типах пользователей электронных таблиц и о том, как классифицируются таблицы. Вы, скорее всего, даже и не думали об этих очень простых вопросах. Поскольку вы обратили свое внимание именно на эту книгу, то важно понять, например, зачем нужны электронные таблицы (если, конечно, вы действительно хотите стать "серьезным программистом"). Вначале остановимся на понятии *приложения электронных таблиц*. Ведь желанным результатом ваших усилий на ниве серьезного программирования наверняка будет именно такое приложение.

Основные характеристики

Что касается использования электронных таблиц, то в данном случае программированием является процесс создания приложений, в которых вместо традиционного языка программирования (такого, например, как C, Pascal или BASIC) применяется программный код электронных таблиц. Однако и в этом случае, и в традиционном программировании приложения будут использоваться другими пользователями, а не их разработчиками.

В настоящей книге приложение электронных таблиц определяется следующим образом. Приложение электронных таблиц — это один файл или группа связанных файлов электронных таблиц, разработанных таких образом, чтобы пользователь, не являющийся их разработчиком, без особой подготовки выполнил необходимые действия. Согласно этому определению, большинство разработанных вами электронных таблиц, вероятно, такими приложениями не являются. На жестком диске у вас могут быть сохранены десятки или сотни файлов электронных таблиц, но наверняка большинство из них не предназначены для других пользователей.

Ниже приведены характеристики эффективного приложения электронных таблиц.

- Конечный пользователь получает возможность выполнить задание, которое, вероятно, не смог выполнить по-другому.
- Предоставляет эффективное решение проблемы (среда электронных таблиц не всегда является оптимальной структурой управления данными).
- Выполняет только ожидаемые действия. Такое требование, возможно, покажется очевидным, но очень часто именно по этой причине приложения нельзя назвать эффективными.
- Выдает точные результаты и не имеет программных ошибок.
- Для выполнения своей работы использует четкие и эффективные методы и алгоритмы.
- Выявляет ошибки, вызванные своим присутствием в системе, не дожидаясь вмешательства пользователя. Обратите внимание, что такие ошибки и программные ошибки — это не одно и то же. Попытка деления на ноль является ошибкой, которая связана с работой приложения. А то, что такая попытка вовремя не пресечена, является уже программной ошибкой.
- Не позволяет пользователю случайно (или умышленно) удалять или видоизменять важные компоненты.
- Имеет простой и понятный графический интерфейс, поэтому пользователь всегда знает, что же следует делать дальше.
- Формулы, макросы и элементы пользовательского интерфейса хорошо документированы, что предоставляет возможность изменять их в случае необходимости.
- Приложение разработано с учетом того, что его можно просто модифицировать, не прибегая к крупномасштабным изменениям (ведь пользователю рано или поздно потребуется внести изменения).
- Располагает легкодоступной справочной системой, которая предоставляет полезную информацию по основным процедурам.
- Должно быть переносимым и работать в любой компьютерной системе, в которой установлены все необходимые программы (в данном случае — требуемая версия Excel).

Не следует удивляться тому, что приложения электронных таблиц можно создавать для различного применения (и в шаблонах, заполняемых данными, и в довольно сложных приложениях с пользовательскими меню и диалоговыми окнами, причем сами приложения могут даже и не выглядеть как электронные таблицы).

Разработчик и конечный пользователь

Мы часто употребляем термины *разработчик* и *конечный пользователь*. Если вы прочли до этого раздела, то, вероятно, что вы или уже разработчик приложений электронных таблиц, или вполне можете им стать.

Итак, определимся с терминологией. *Разработчик* — это человек, который создает приложение электронных таблиц. В совместных проектах число разработчиков больше одного (команда разработчиков). *Конечный пользователь* (его для краткости будем называть просто *пользователем*) — это человек, применяющий результаты деятельности разработчика по программированию электронных таблиц. Во многих случаях конечных пользователей бывает достаточно много, а разработчиком часто является один из пользователей.

Кто такие разработчики

В течение 15-ти лет я занимаюсь преподаванием методологии разработки. Поэтому чаще всего имеют дело с теми, кто называет себя разработчиками электронных таблиц. Среди них различаются две основные группы.

- *Внутренние специалисты*, которые тесно сотрудничают с пользователями и основательно знают их потребности. Во многих случаях эти разработчики также являются пользователями своего приложения. Часто бывает так, что они разрабатывают приложение, чтобы решить только одну конкретную проблему.
- *Специалисты со стороны*, как правило, приглашаются с целью решить проблему. В большинстве случаев такие разработчики знакомы с вопросом лишь в общих чертах, однако хорошо знают специфику приложения, которое разрабатывают. Разработчики могут быть сотрудниками той же самой компании, которой требуется приложение, но только другого подразделения.

У некоторых специалистов на разработку приложения уходит все рабочее время. Они могут быть как внутренними специалистами, так и специалистами со стороны. Достаточно много консультантов (со стороны) неплохо зарабатывают, работая "свободными художниками" по созданию приложений электронных таблиц.

Другие же разработчики электронных таблиц не посвящают этому все свое рабочее время и даже не осознают, что они разрабатывают соответствующие приложения. Такими разработчиками часто выступают работающие в офисах компьютерные "знатоки", которые, кажется, все знают о компьютерах и программах. Часто эти люди создают приложения электронных таблиц как раз для того, чтобы облегчить себе жизнь. Ведь время, которое они тратят, чтобы разработать для других сотрудников хорошее приложение, часто экономит часы, которые пришлось бы потратить на обучение специалистов. Кроме того, время разработки такого приложения значительно меньше времени, которое требуется, чтобы отвечать на вопросы сотрудников.

Разработчики электронных таблиц обычно принимают участие в следующих операциях, самостоятельно выполняя большинство из них или даже все.

- Определение потребностей пользователя.
- Планирование приложения, которое соответствует этим потребностям.
- Определение наиболее подходящего интерфейса пользователя.
- Создание электронной таблицы, формул, макросов и пользовательского интерфейса.
- Тестирование приложения в разных условиях.
- Изменение приложения с целью повысить его надежность и отказоустойчивость (часто по результатам тестирования).

- Обеспечение эстетической привлекательности и наглядности приложения.
- Документирование усилий, потраченных на разработку.
- Размещение приложения в компьютере пользователя.
- Обновление приложения в случае необходимости.

Более подробно об этих операциях рассказано в главе 6.

Разработчики должны быть хорошо знакомы со средой, в которой они работают (в данном случае это программа Excel). Конечно, по уверениям компании Microsoft, использовать эту программу очень легко, но определение легкости отличается для каждого конкретного пользователя. Для разработки с помощью Excel нестандартных приложений электронных таблиц требуется глубокое знание формул, функций, макросов, пользовательских диалоговых окон, пользовательских панелей инструментов, а также надстроек и команд меню. Большинство пользователей, как правило, не соответствуют выдвигаемым требованиям и не имеют намерения изучать эти подробности. Итак, перейдем к следующей теме — классификации пользователей электронных таблиц.

Классификация пользователей электронных таблиц

Пользователи, работающие с электронными таблицами (как пользователи-разработчики, так и обычные пользователи) различаются по двум критериям: *степени* или *опытности* использования электронных таблиц и *интересу к изучению* самых таблиц.

Каждый из этих критериев имеет три уровня. Комбинируя уровни обоих параметров, получаем девять вариантов, которые представлены в табл. 5.1. Однако рассматривать мы будем только семь из них. Дело в том, что *минимальный* интерес к электронным таблицам обычно проявляют те пользователи, которые имеют умеренный и очень высокий опыт работы с ними (интерес как раз и стимулировал таких пользователей к приобретению опыта). Что же касается пользователей, которые обладают большим опытом работы с электронными таблицами и низким уровнем интереса, то из них обычно получаются не очень хорошие разработчики.

Таблица 5.1. Классификация пользователей электронных таблиц по опытности и интересу

Уровень опыта	Интерес отсутствует	Умеренный интерес	Очень большой интерес
Малый опыт	Пользователь	Пользователь	Пользователь/Потенциальный разработчик
Умеренный опыт	-	Пользователь	Разработчик
Очень большой опыт	-	Пользователь	Разработчик

Очевидно, что у разработчиков электронных таблиц должен быть как немалый *опыт* работы с этими таблицами, так и высокий к ним *интерес*. Те, у кого небольшой опыт работы, но имеется интерес, являются потенциальными разработчиками. Все, что им необходимо, — это приобретение опыта. И если вы читаете эту книгу, то, вероятно, относитесь к одной из категорий последнего столбца этой таблицы.

Пользователи приложений электронных таблиц

Оставшиеся ячейки последней таблицы соответствуют тем конечным пользователям электронных таблиц, которых вы считаете “потребителями” приложений электронных таблиц. Когда вы разрабатываете такое приложение, предназначенное для других людей, то вам необходимо знать, какие из этих групп пользователей действительно будут его применять.

Значительная часть пользователей не имеют опыта и интереса. Это люди, которым электронная таблица необходима для работы. Она рассматривается просто как средство для достижения конечной цели. Обычно таким пользователям мало известно о компьютерах и программах, и, как правило, им неинтересно изучать то, что не касается выполнения их работы. Возможно, компьютеры их немного пугают. Очень часто им даже неизвестна версия того процессора электронных таблиц, которую они используют, кроме того, они не знают всех возможностей программы. Очевидно, что приложения, разработанные для этой группы, должны быть дружественны к пользователям, т.е. простые, не внушающие страх, легкие в использовании и по возможности отказоустойчивые.

С точки зрения разработчика, более интересной группой являются те пользователи, которые обладают умеренным опытом работы с электронными таблицами и которые заинтересованы в том, чтобы знать больше. Эти пользователи имеют понятие о формулах, пользуются функциями надстроек и обычно знают о возможностях программного продукта. Они, как правило, ценят тот труд, который вы вложили в приложение, на них часто производят впечатление приложенные вами усилия. Более того, эти пользователи часто предлагают прекрасные идеи, которые помогают улучшить ваш продукт. Приложения, разработанные для этой группы, также должны быть дружественны к пользователям (легкие в использовании и отказоустойчивые), но это еще не все. Они, кроме того, могут быть более сложными и уникальными, чем приложения, которые предназначены для аудитории, состоящей из менее опытных и заинтересованных пользователей.

Решение проблем с помощью процессора электронных таблиц

Выше речь шла о таком базовом понятии, как приложение электронных таблиц; вы узнали о некоторых типах конечных пользователей и разработчиков приложений, выяснили, почему люди вообще используют процессоры электронных таблиц. Теперь настало время показать, какие задачи решаются с помощью приложений электронных таблиц.

Вы, возможно, уже имеете достаточно хорошее представление о тех задачах, для решения которых он применяется. Традиционно такие программы использовались в тех приложениях, которые по своей природе являются в значительной степени *интерактивными*. Хороший пример таких приложений — бюджет корпорации. После подготовки соответствующей модели (т.е. создания расчетных формул) управление бюджетом сводится к введению конечных значений и изучению итогов, полученных в результате автоматических вычислений. Часто разработчикам приложения расчета бюджета достаточно распределить фиксированные ресурсы по разным видам деятельности и представить результаты в привлекательном (или, как минимум, удобочитаемом) виде. Конечно же, процессор электронных таблиц является для этого идеальным средством.

Впрочем, задачи, аналогичные описанной, составляют лишь небольшой процент того, для чего разрабатываются электронные таблицы. Зачастую пользователи процессоров электронных таблиц (особенно в последние годы) применяют эти программы для решения не тех задач, для которых они предназначались изначально.

Приведем несколько примеров нетрадиционного применения процессора электронных таблиц Excel.

- *В качестве средства проведения презентаций.* Например, используя исключительно Excel, вы можете с минимальными усилиями создать привлекательное интерактивное слайд-шоу, выводимое на экран монитора.

- *Как инструмент ввода данных.* Программа электронных таблиц часто является самым эффективным средством решения таких задач, как повторный ввод данных. Введенные данные могут экспортироваться в самые разные форматы, применяемые другими программами.
- *В качестве генератора бланков.* Многим пользователям для создания привлекательных печатных бланков проще обратиться к инструментам форматирования Excel, а не изучать настольную издательскую систему, например, PageMaker.
- *Как текстовый процессор.* Функции управления текстом, которые присутствуют во всех процессорах электронных таблиц, предоставляют возможность манипулировать текстом так, как это невозможно даже в текстовом процессоре.
- *В качестве платформы простых игр.* Конечно, разработчики программы Excel о таком ее применении и не думали. Однако я загрузил из Internet (а также написал собственноручно) интересные стратегические игры, в которых применяются инструменты Excel и других процессоров электронных таблиц.

Вы, вероятно, можете пополнить этот список, вспомнив еще многие другие примеры.

Ирония в том, что универсальность процессоров электронных таблиц является палкой о двух концах. С одной стороны, появляется искушение применить такую программу для решения любой возникшей проблемы. А с другой — вы часто выбиваетесь из сил, пытаясь с помощью электронных таблиц справиться с проблемой, для которой легче найти другое решение.

Основные типы электронных таблиц

В этом разделе приведена классификация электронных таблиц, включающая нескольких основных типов. Она проиллюстрирует, каким образом электронные таблицы вписываются в общую картину управления данными с помощью компьютера. Конечно, данная классификация является довольно условной. Она создана исключительно на основе моего личного опыта. Более того, ее категории часто пересекаются, однако к ним относится большинство электронных таблиц.

Предложим такие названия категориям (типам) электронных таблиц:

- электронные таблицы “на скорую руку”;
- электронные таблицы “не для посторонних глаз”;
- однопользовательские приложения;
- приложения-“спагетти”;
- приложения-утилиты;
- надстройки с функциями рабочих листов;
- одноблоковые бюджеты;
- модели “что-если”;
- приложения для хранения и доступа к данным;
- клиентские приложения доступа к базам данных;
- приложения “под ключ”.

О каждой из этих категорий рассказывается в следующих разделах.

Электронные таблицы “на скорую руку”

Вероятно, это самый распространенный тип электронных таблиц. Большинство электронных таблиц из этой категории являются небольшими. Они разрабатываются для того, чтобы быстро решить проблему или получить ответ на вопрос. Рассмотрим пример. Вы собираетесь купить новую машину, поэтому необходимо для разных сумм кредита вычислить размер ежемесячной выплаты. Или вы решили создать диаграмму, которая покажет объем продаж вашей компании по месяцам. Введите 12 значений, затем скопируйте диаграмму и вставьте в документ, создаваемый в текстовом процессоре.

В обоих случаях на разработку модели вы, похоже, тратите несколько минут, и, конечно же, у вас нет времени документировать свои действия. Скорее всего, вы и не думаете создавать какие-либо макросы или пользовательские диалоговые окна. Наверняка вы даже не считаете нужным сохранять эти простые электронные таблицы на диске. Поэтому электронные таблицы этой категории не являются приложениями.

Электронные таблицы “не для посторонних глаз”

Как следует из названия, электронные таблицы, которые попадают в эту категорию, не увидит и не будет использовать никто, кроме вас — их разработчика. В качестве примера можно привести файл с информацией, которая относится к оплате налогов, начисляемых на основе ваших доходов. Вы открываете свой файл, когда к вам по почте приходит чек, или вы “влезаете” в расходы, которые можно рассматривать как производственные, или же покупаете у торговцев на улице ворованную автомагнитолу и т.д. Другим примером является электронная таблица, в которой вы ведете учет времени, потраченного вашими сотрудниками (отсутствие на работе по болезни, отпуск и т.д.) попусту или попросту прогулянного.

Электронные таблицы данной категории отличаются, например, от созданных “на скорую руку”, которые не являются одноразовыми, поэтому их и сохраняют в файлах. Однако на них не стоит тратить много времени — примените простое форматирование (лишь в случае необходимости). В электронных таблицах этого типа также отсутствуют инструменты обнаружения ошибок: вы знаете, каким образом формулы создавались, поэтому вам хорошо известно, как избежать ввода данных, приводящих к ошибочным результатам. При появлении ошибки вы сразу будете знать, чем она вызвана.

Сложность электронных таблиц этой категории со временем возрастает, однако они не являются приложениями. Предположим, у вас есть рабочая книга Excel, в которой вы ведете учет своих доходов по источникам их поступления. Эта книга на время создания была довольно простой, но вы регулярно пополняете ее новыми элементами: сводными формулами, улучшенным форматированием и даже диаграммой доходов по месяцам. Самым последним таким изменением в диаграмме стала линия прогнозирования, предназначенная для того, чтобы планировать доходы, основываясь на трендах за прошедшие периоды. Вероятно, вы будете и в дальнейшем улучшать этот файл, и тогда он сможет перейти в категорию однопользовательских приложений.

Однопользовательские приложения

Приложения электронных таблиц, используемые только их разработчиком, однако по своей сложности вышедшее далеко за пределы электронных таблиц “не для посторонних глаз”. Например, я разработал рабочую книгу, чтобы вести в ней учет зарегистрированных пользователей условно-бесплатных приложений. Книга начинается простой базой данных, расположенной на одном рабочем листе (она предназначается только для просмотра разработчиком). Однако вскоре эта рабочая книга также понадобилась для создания накладных и почтовых этикеток. Потратив однажды около часа на создание макросов, понял, что превратил эту рабочую книгу из приложения “не для посторонних глаз” в настоящее однопользовательское приложение.

Несмотря на то, что никто другой никогда не воспользуется этой электронной таблицей, она все равно стала симпатичным маленьким приложением, которым довольно легко пользоваться. Что же касается разработки, то время, которое было потрачено на доведение этой электронной таблицы до уровня однопользовательского приложения, не прошло даром, так как в результате сэкономлено несколько часов работы. Теперь в этом приложении вы найдете кнопки для запуска макросов, поэтому потребуется намного меньше усилий, чтобы справиться с учетом всех клиентов; вы также сэкономите время, работая с почтовыми программами.

Создавая для себя однопользовательские приложения, вы имеете прекрасную возможность попрактиковаться с инструментами, которыми пользуются разработчики Excel-приложений. Например, вы можете научиться создавать пользовательские диалоговые окна, видоизменять меню, создавать пользовательскую панель инструментов, писать VBA-макросы и т.д. Работа над содержательным проектом (даже если он содержательный только для вас) — это эффективный способ изучать сложные функции Excel (как, впрочем, и любой другой программы).

Приложения-"спагетти"

Среди электронных таблиц все еще распространены *приложения-"спагетти"*. Само понятие возникло по причине того, что в отдельных частях приложения разобраться бывает довольно трудно; эти части во многом так же спутаны между собой, как спагетти на тарелке. Большинство этих электронных таблиц разрабатывались как специализированные однопользовательские приложения. Но со временем они перешли во владение других пользователей, которые внесли свои изменения. По мере того, как требования менялись, а сотрудники приходили и уходили, одни части появлялись, а другие игнорировались. Спустя некоторое время, первоначальное назначение рабочей книги забылось. В результате получился файл, используемый довольно часто, однако никто по-настоящему не знает, как же этот файл в точности работает.

Каждый, кто имеет дело с приложениями-"спагетти", знает, что их необходимо полностью переделать. Но так как никто не знает, как это сделать, со временем дела обстоят все хуже и хуже. Консультанты по процессорам электронных таблиц зарабатывают немалые деньги, занимаясь распутыванием таких приложений. Как правило, в процессе улучшения приложений-"спагетти" наиболее эффективным является следующее решение — заново определить, что же требуется пользователям, и с самого начала создать новое приложение.

Приложения-утилиты

Никто никогда еще не был полностью доволен используемым процессором электронных таблиц. И какой хорошей бы ни была программа Excel, в ней все равно находят недостатки. Поэтому перейдем к следующей категории электронных таблиц — *приложениям-утилитам*. Утилиты — это специальные инструменты, которые предназначены для выполнения единственной повторяющейся задачи. Например, если вы часто занимаетесь импортом текста в Excel, то вам, возможно, требуются специальные команды для обработки текста, в частности, для преобразования (без использования формул) выделенного текста в верхний регистр. В данном случае советуем разработать утилиту для обработки текста, которая будет выполнять именно то, что вам необходимо.

НА КОМПАКТ-ДИСКЕ

Power Utility Pak — это коллекция приложений-утилит для программы Excel. Они разработаны для того, чтобы расширить возможности программы. Эти утилиты работают, как обычные команды Excel. На прилагаемом к книге компакт-диске вы найдете условно-бесплатную версию этого пакета. При желании можно заказать.

По своей природе приложения-утилиты являются достаточно универсальными. Что касается макросов, то многие из них предназначены для того, чтобы выполнять конкретную операцию с данными конкретного типа, расположенными в рабочей книге (опять же) конкретного типа. Эффективное приложение-утилита работает подобно тому, как и обычная команда Excel. Другими словами, утилите необходимо распознать контекст, в котором должна выполняться команда, и выполнить соответствующее действие. Чтобы утилита умела обработать любые возможные ситуации, обычно используется громоздкий код, предназначенный для обработки ошибок.

В приложениях-утилитах всегда используются макросы, в них также могут применяться пользовательские диалоговые окна. К счастью, создать такие утилиты с помощью Excel довольно легко: их следует преобразовать в надстройки и присоединить к пользовательскому интерфейсу Excel, чтобы они выглядели, как часть программы.

Тема создания утилит является настолько важной, что ей посвящена целая глава. В главе 16 рассказывается о том, как создавать на VBA пользовательские утилиты Excel.

Надстройки с функциями рабочих листов

Как вы знаете, Excel располагает огромным количеством функций рабочих листов, которые можно использовать в формулах. Но случается, что вам необходима определенная функция, а ее поиски дают отрицательный результат. В таком случае создайте свою собственную функцию, используя для этого VBA. Благодаря пользовательским функциям рабочих листов формулы часто становятся проще, а поддерживать электронные таблицы — легче.

В главе 10 изложена полная информация о создании пользовательских функций рабочих листов, а также приведены соответствующие примеры.

Одноблоковые бюджеты

Под *одноблоковым бюджетом* подразумеваем рабочий лист (не обязательно модель бюджета), который состоит из одного блока ячеек. Верхняя его строка может быть составлена из имен, относящихся к промежуткам времени (месяцам, кварталам или годам), а левый столбец обычно состоит из категорий определенного типа. Как правило, нижняя строка и правый столбец составлены из итоговых формул. В блоке ячеек могут использоваться формулы подсчета промежуточных итогов.

Данный тип электронных таблиц очень распространен. С учетом как раз этой модели была разработана программа VisiCalc (самый первый процессор электронных таблиц). В большинстве случаев простые модели одноблоковых бюджетов не являются удачными кандидатами в приложения, так как они слишком просты. Впрочем, среди них *существуют* исключения. Например, вы могли бы подумать над превращением такой электронной таблицы в приложение, если моделью одноблокового бюджета является громоздкая трехмерная электронная таблица, в которую необходимо включить сводные данные из других файлов или которой будут пользоваться руководители отделов, возможно, не разбирающиеся в электронных таблицах.

Модели “что-если”

Многие считают модель “что-если” воплощением всего самого лучшего, что имеется в электронных таблицах. Способность мгновенно пересчитывать тысячи формул делает процессоры электронных таблиц идеальным инструментом для финансового моделирования, а также для других моделей, которые зависят от значений нескольких переменных. Если подумать, то почти каждая электронная таблица с формулами является моделью “что-если” (она часто распространяется в виде шаблона). Изменение значения в ячейке, используемой в формуле, представляет ситуацию постановки вопроса “Что будет, если...?”. Кроме того, приложения данной категории довольно сложные. Они состоят из тех электронных таблиц, которые специально разрабатывались для прогнозирования ситуаций влияния отдельных значений на конечный результат.

Модели “что-если” являются хорошими кандидатами в приложения, ориентированные на пользователя, особенно если модель будет использоваться продолжительное время. При условии создания для приложения удачного графического интерфейса его сможет легко использовать даже тот, кто совсем не разбирается в компьютерах. Например, вы бы могли создать интерфейс, который предоставляет пользователю возможность задавать имена для различных наборов условий, мгновенно просматривать результаты выбранного сценария и одним щелчком на кнопке создавать правильно отформатированную сводную диаграмму.

Электронные таблицы для хранения данных и доступа к ним

Не удивительно, что электронные таблицы часто используются для хранения списков или для выполнения скромных манипуляций с базами данных. Многие пользователи считают, что просматривать данные и манипулировать ими намного легче в электронной таблице, а не с помощью процессора баз данных. Начиная с Excel 97, в каждой электронной таблице находится 65 536 строк. Увеличение размера рабочей области значительно расширило потенциальные возможности по управлению базами данных.

Электронные таблицы этой категории часто являются кандидатами в приложения, особенно если конечным пользователям необходимо выполнять операции умеренной сложности. Впрочем, встроенное в Excel диалоговое окно формы данных и его команды автофильтрования делают работу с базами данных такой легкой, что даже начинающие пользователи могут быстро научиться выполнять в базах данных простые операции.

Что же касается сложных приложений баз данных, в частности, таких, в которых используется огромное количество таблиц с заданными связями между ними, то для них больше подходит настоящий процессор баз данных — например, Access.

Клиентские программы баз данных

Электронные таблицы все чаще применяются для доступа к внешним базам данных. Пользователи электронных таблиц с помощью инструментов Excel получают доступ к данным, хранящимся во внешних файлах, даже если у этих данных разный формат. Когда вы создаете приложение, которое выполняет эту задачу, то к нему иногда обращаются как к управленческой информационной системе (*executive information system — EIS*). Такая система комбинирует данные из нескольких источников и сводит их для пользователей.

Доступ из электронной таблицы к внешним базам данных часто вызывает страх у начинающих пользователей. Создание управленческой информационной системы является идеальным способом применения Excel, так как основная цель подобных систем обычно состоит в том, чтобы обеспечить простоту в использовании.

Приложения "под ключ"

Последняя категория электронных таблиц является самой сложной. Говоря *"под ключ"*, подразумеваем такую готовность к использованию, когда конечный пользователь должен иметь минимальную подготовленность или вообще не обладать ею. Например, при загрузке файла появляется окно, позволяющие сделать совершенно однозначный выбор. Приложения "под ключ" могут выглядеть так, будто они созданы не с помощью процессора электронных таблиц, и часто пользователь взаимодействует не с ячейками, а с диалоговыми окнами.

Электронные таблицы многих описанных выше категорий вполне можно сделать приложениями "под ключ". Среди общих элементов таких приложений самыми главными являются хорошее планирование, обработка ошибок и система пользовательского интерфейса. О них речь пойдет в следующих главах этой книги.

Глава 6

Принципы разработки приложений электронных таблиц

В ЭТОЙ ГЛАВЕ...

В данной главе будут приведены *общие* правила, которые вы, возможно, сочтете полезными, когда будете учиться создавать эффективные приложения, работающие в Excel.

- ♦ Основные действия по разработке приложения электронных таблиц
- ♦ Определение потребностей конечного пользователя и проектирование приложений, соответствующих этим потребностям
- ♦ Правила разработки и тестирования приложений
- ♦ Документирование усилий по разработке приложения и создание пользовательской документации

К сожалению, не существует такого простого и безошибочного метода, который бы гарантировал создание эффективного приложения электронных таблиц. У каждого разработчика собственный стиль создания таких приложений, и "наилучший способ" определяет для себя сам разработчик. Кроме того, каждый проект, за который вы беретесь, будет отличаться от других, и поэтому для его реализации потребуется особый подход. И наконец, требования и общие представления людей, с которыми (или на которых) вам предстоит работать, также играют определенную роль в процессе разработки.

Как уже отмечалось в предыдущей главе, разработчики электронных таблиц несут ответственность за выполнение следующих требований.

- ♦ Определение потребностей пользователя.
- ♦ Планирование приложения, которое соответствует этим условиям.
- ♦ Разработку наиболее подходящего интерфейса пользователя.
- ♦ Создание электронной таблицы, формул, макросов и пользовательского интерфейса.
- ♦ Тестирование приложения в разных условиях.
- ♦ Изменение приложения с целью повышения его надежности и отказоустойчивости (часто по результатам тестирования).
- ♦ Эстетическую привлекательность и наглядность приложения.
- ♦ Документирование усилий, потраченных на разработку.
- ♦ Размещение приложения в компьютере пользователя.
- ♦ Обновление приложения в случае необходимости.

Не обязательно выполнять все эти требования при создании каждого приложения, да и порядок их выполнения в разных проектах может быть разным. Перечисленные действия описаны в следующих разделах. Что же касается технических деталей, то в большинстве случаев их также можно найти в последующих главах.

Определение потребностей пользователя

Когда вы приступаете к разработке проекта приложения электронных таблиц, то одним из первых действий является точное определение потребностей конечного пользователя. И если вы не сможете заранее оценить потребности аудитории, то позднее это обернется дополнительной работой по устранению недостатков. Именно поэтому оценку потребностей необходимо совершить в самую первую очередь.

В отдельных случаях вы можете быть близко знакомы с конечными пользователями, и даже сами можете выступать одним из них. Что же касается остальных случаев (например, вы работаете консультантом, который разрабатывает проект для нового клиента), то о пользователях или об их запросах вы, возможно, вообще ничего не будете знать.

Ниже приведены основные правила, придерживаясь которых вы облегчите начальную фазу разработки.

- Не убеждайте себя в том, что заранее знаете потребности пользователей. Если на этом этапе основываться на предположениях, то позднее гарантировано возникновение проблем.
- Если существует такая возможность, то говорите непосредственно с потенциальными клиентами приложения, а не только с руководителем проекта или управляющим фирмы.
- Узнайте, что уже сделано (если только сделано) для удовлетворения потребностей пользователей. Вы, возможно, сэкономите часть своего времени, переделав уже существующее приложение. В крайнем случае, изучая уже имеющиеся решения, вы более подробно ознакомитесь с работой конечных пользователей.
- Определите, какие ресурсы имеются в распоряжении пользователя. Постарайтесь, например, узнать, существуют ли какие-либо аппаратные или программные ограничения, которые нужно учитывать.
- Если можно, определите все типы систем, в которых будет запускаться ваше приложение. В том случае, если приложение будет выполняться в низкопроизводительных системах, примите этот факт к сведению.
- Узнайте, какая версия (или версии) Excel используется в системе. Конечно, компания Microsoft делает все возможное, чтобы убедить пользователей применять самые последние версии тех программ, которые она выпускает. Однако недавно были опубликованы результаты опроса, согласно которому самые последние версии пакета Microsoft Office имеет менее половины всех пользователей этого пакета.
- Узнайте уровень квалификации конечных пользователей. Эта информация поможет вам правильно разработать приложение.
- Определите, как долго будет использоваться приложение и ожидаются ли изменения в нем в будущем. Знание этого вопроса окажет влияние на выполняемые операции и поможет спланировать изменения. Ну а как определить потребности пользователя? Если вас попросят разработать приложение электронных таблиц, то рекомендуется встретиться с конечным пользователем и задать интересующие вас вопросы. А еще лучше, если вы законспектируете полученные

ответы, затем создадите алгоритм работы приложения, обратите внимание на второстепенные детали и сделаете все необходимое для того, чтобы разрабатываемый вами продукт был именно таким, каким вы ожидаете его увидеть.

И последнее замечание. Не удивляйтесь, если назначение проекта за время его разработки успеет измениться. Такая ситуация встречается довольно часто, и если изменения не окажутся для вас сюрпризом, а вы к ним, наоборот, будете *готовы*, то всегда окажетесь в более выигрышной позиции. На всякий случай проверьте, что в вашем контракте (если таковой имеется) изменения спецификаций проекта учтены в вашу пользу.

Проектирование приложения, соответствующего потребностям пользователей

Определив потребности конечных пользователей, вы можете почувствовать желание погрузиться в работу, т.е. прийти на помощь тому, кто уже страдает от нерешенной проблемы. Но постарайтесь набраться терпения. Строители не возводят дом без набора чертежей, да и вам не следует разрабатывать приложение электронных таблиц без определенного рода плана. Конечно, правильность плана зависит от области действия проекта и привычного вам стиля работы, однако подумайте хотя бы *некоторое* время над тем, что же вы собираетесь делать и чего хотите достичь.

Вам не повредит, если, прежде чем закатать рукава и сесть за клавиатуру, вы обдумаете разные способы, которые помогут приблизить решение проблемы. Вот здесь как раз и окупится глубокое познание возможностей Excel! Всегда лучше обходить незнакомые закоулки стороной, чем потом блуждать в них.

Если вы попросите десять "гуру" Excel разработать приложение на основе очень точных спецификаций, то, скорее всего, получите десять разных реализаций проекта, которые все, как одна, будут соответствовать предложенным спецификациям. Среди этих решений некоторые будут определенно лучше, чем другие, поскольку Excel позволяет выполнить одно и то же задание несколькими способами. И если вы досконально знаете эту программу, то будете иметь достаточно хорошее представление о методах, которые существуют в вашем распоряжении, поэтому сможете выбрать самый оптимальный спосом для решения текущего проекта. Часто бывает так, что благодаря только творческому подходу рождается наиболее эффективный способ, значительно превосходящий другие методы.

Итак, на начальном этапе планирования проекта вам придется обдумать следующие вопросы.

- *Файловая структура.* Подумайте над тем, что вы будете использовать: одну рабочую книгу с множеством листов, несколько однолистных рабочих книг или файл шаблона.
- *Структура данных.* Обязательно учтите структуру данных, которые будут использоваться в приложении. В том числе необходимо решить, использовать файлы внешних баз данных или хранить всю информацию в рабочих листах.
- *Формулы или VBA.* Обдумайте, что требуется для проведения вычислений: использование формул или написание процедур VBA? Каждый из представленных вариантов имеет свои достоинства и недостатки.
- *Надстройка или файл XLS.* В некоторых случаях лучшим вариантом конечного продукта является надстройка, хотя не исключено применение надстройки вместе со стандартной рабочей книгой.

- *Версия Excel.* Где будет запускаться ваше Excel-приложение: в Excel 2003, Excel 2000 или Excel 97? Ну а как насчет Excel 95 и Excel 5? Будет ли оно работать на платформе Macintosh? Это очень важные вопросы, поскольку в каждую новую версию Excel добавляются такие возможности, которые отсутствуют в предыдущих.
- *Как обрабатывать ошибки.* Для приложений немаловажным вопросом является обработка ошибок. Определите, как ваше приложение будет "отлавливать" ошибки, и что потом оно будет с ними делать. Например, если ваше приложение применяет форматирование к активному рабочему листу, то необходимо предусмотреть случай, когда активным будет лист диаграмм.
- *Использование специальных возможностей.* Если в вашем приложении будет суммироваться большое количество данных, то подумайте над использованием такого средства Excel, как сводные таблицы. Вам также потребуется такая возможность Excel, как проверка данных, чтобы тестировать вводимые данные на правильность.
- *Вопросы производительности.* Решить вопрос увеличения производительности и эффективности вашего приложения следует еще на стадии проектирования, а не тогда, когда создание приложения завершено и от пользователей поступают жалобы.
- *Уровень безопасности.* Как вы, возможно, знаете, в Excel предусмотрено несколько вариантов защиты, которые призваны предотвратить доступ к определенным элементам рабочей книги. Например, вы можете блокировать ячейки, чтобы нельзя было изменить находящиеся в них формулы, или назначить пароль, чтобы неавторизованные пользователи не смогли просматривать определенные файлы или получать доступ к ним. Вы облегчите свою работу, если заблаговременно и точно определите, какой именно компонент нуждается в защите и каков должен быть уровень этой защиты.



Не стоит полностью полагаться на средства защиты данных Excel. Если вам нужно абсолютно защищенное приложение, то Excel вряд ли для этого подойдет.

На данном этапе вам, возможно, придется иметь дело со многими другими трудностями разработки приложения. Важно рассмотреть все возможности и не приниматься за первое решение, которое придет вам в голову.

Кроме того, при разработке приложения не забывайте о его возможных изменениях. Вы добьетесь больших успехов, если ваше приложение будет максимально общим. Например, не создавайте процедуру, которая работает только с определенным диапазоном ячеек. Пусть ваша процедура в виде аргумента принимает любой диапазон данных. Когда придет время изменений в проекте, такая возможность существенно облегчит процесс редактирования приложения. Кроме того, вы, возможно, увидите, что текущий проект в определенной мере напоминает другой проект. Поэтому при планировании всегда помните о такой возможности, как повторное использование одних и тех же структур.

Обучение в процессе разработки

Теперь несколько слов о реальном положении вещей. Excel — программа изменчивая. Период между ее обновлениями составляет от 18 до 24 месяцев. Это означает, что в вашем распоряжении менее двух лет, чтобы справиться с текущими инновациями, а иначе вам придется бороться не только с ними, но и с другими нововведениями.

Программа Excel 5, которая подарила нам VBA, стала для разработчиков Excel "изменением парадигмы" управления данными. Ранее тысячи людей зарабатывали себе на жизнь, создавая

приложения Excel, которые, в основном, писались на макроязыке XLM, представленном в Excel 2, 3 и 4. Начиная с Excel 5, стали доступны десятки новых инструментов, активно используемых разработчиками.

Появление программы Excel 97 заставило разработчиков столкнуться с еще одной “сменой парадигмы”. В этой версии появился новый формат файлов, редактор языка Visual Basic, и пользовательские формы, которые пришли на замену диалоговым листам. Excel 2000 и Excel 2002 также обеспечили дополнительные возможности, но эти изменения уже не были столь радикальными, как изменения в предыдущих версиях.

После Excel 2003 мы, вероятно, будем свидетелями еще одной значительной “смены парадигмы”. В настоящее время компания Microsoft работает над совершенствованием технологии .NET, и похоже на то, что пакет Office станет частью этой технологии. Как утверждают в кругах разработчиков, на смену VBA придет VSA (Visual Studio for Applications).

Язык VBA изучать нетрудно, но потребуется время, чтобы вы почувствовали себя с ним комфортно; для совершенного же овладения этим языком придется приложить немало усилий. Развитие VBA продолжается. Следовательно, не удивительно, что, разрабатывая с помощью VBA приложения, вы одновременно изучаете этот язык. Скажем более — изучить VBA невозможно, если вплотную не заниматься разработкой приложений. Намного легче изучать VBA, имея проект, для реализации которого требуется использование только VBA. Изучение языка VBA только с целью его изучения к значимым результатам вряд ли приведет.

Опыт показывает, что нельзя при решении проблемы целиком полагаться на конечных пользователей. Предположим ситуацию, когда вы встречаетесь с руководителем, и он говорит, что его отделу требуется приложение, генерирующее текстовые файлы, которые будут импортироваться в другое приложение. В данном случае главное — не путать потребности пользователя с искомым решением. Пользователю необходим совместный доступ к данным. А использование промежуточного текстового файла является всего лишь частным решением этой проблемы. Ведь могут быть и другие частные решения — например, прямая передача информация с помощью DDE или OLE. Иначе говоря, не позволяйте пользователю представлять его проблему в виде конечной задачи. Определить наиболее удачное решение *общей* проблемы — это уже *ваша* работа.

Определение удобного пользовательского интерфейса

Разрабатывая электронные таблицы, которые будут использоваться другими людьми, вы должны обратить особое внимание на пользовательский интерфейс. *Пользовательский интерфейс* — это метод взаимодействия пользователя с приложением — щелчки на кнопках, использование меню, нажатие клавиш, доступ к панелям инструментов и т.д.

Не забывайте также о конечном пользователе. Вероятно, у вас намного больше опыта работы с компьютерами, чем у конечных пользователей, а интерфейс, который является наглядным для вас, для остальных может и не быть таковым.

Один из способов обеспечить удобный пользовательский интерфейс — это положиться на встроенные возможности программы Excel: имеющиеся меню, панели инструментов, панели прокрутки и т.д. Другими словами, вы можете создать рабочую книгу и затем предоставить пользователю возможность сделать с ней все, что ему захочется. Такое решение, возможно, является оптимальным, но только тогда, когда приложение предназначено для тех, кто хорошо знает Excel. Впрочем, чаще всего обнаруживается, что аудитория вашего приложения состоит из относительно неопытных (и часто незаинтересованных) пользователей. Это затрудняет вашу работу и заставляет уделять особое внимание пользовательскому интерфейсу, который предназначен для управления приложением.

В Excel содержится несколько средств, необходимых при разработке пользовательского интерфейса:

- пользовательские диалоговые окна (пользовательские формы);
- элементы управления (такие, например, как `ListBox` (поле со списком) и `CommandButton` (командная кнопка)), размещаемые непосредственно на рабочем листе;
- пользовательские меню;
- пользовательские панели инструментов;
- пользовательские комбинации клавиш.

Эти средства будут кратко рассмотрены в следующих разделах, а более подробно — в следующих главах.

Создание пользовательских диалоговых окон

Если вы какое-то время работали в Excel, то несомненно знакомы с диалоговыми окнами. Пользовательские диалоговые окна играют важную роль в тех интерфейсах, которые вы разрабатываете для своих приложений.

Excel версии 97 принесла с собой полностью новый способ создания пользовательских диалоговых окон. Речь идет о пользовательских формах (`UserForm`). Впрочем, диалоговые листы, разработанные в Excel 5/95, поддерживаются и в последующих версиях программы. Что же касается этой книги, то в ней внимание уделяется исключительно пользовательским формам.

С помощью пользовательского диалогового окна пользователь может ввести те или иные данные, получить выбранные им варианты или предпочтения, а также задать ход выполнения всего приложения. Такие диалоговые окна хранятся в пользовательских формах (одно окно на форму). Создавать и редактировать пользовательские диалоговые окна вы можете в редакторе Visual Basic (Visual Basic Editor — VBE), доступ к нему вы получите с помощью комбинации клавиш <Alt+F11>. Элементы, из которых состоит диалоговое окно, называются *элементами управления* (к ним относятся кнопки, раскрывающиеся списки, флажки и т.д.). Они также называются *элементами управления ActiveX*. Можно пользоваться не только стандартным набором таких элементов, который поддерживается в Excel. Вы вправе вставлять элементы управления, которые созданы сторонними производителями.

Добавив элемент управления в диалоговое окно, можно связать его с ячейкой рабочей таблицы, поэтому ему не потребуется макрос для управления (если не считать простого макроса, с помощью которого отображается само диалоговое окно). Связать элемент управления с ячейкой несложно, однако такой способ приема данных, вводимых пользователем в диалоговое окно, не всегда является лучшим. Гораздо чаще для работы с созданными вами диалоговыми окнами придется создавать VBA-макросы.

Подробно о пользовательских формах рассказывается в части IV.

Использование элементов управления ActiveX в рабочем листе

В Excel элементы управления ActiveX, предназначенные для пользовательских форм, можно также вставлять и на графический слой. На рис. 6.1 представлена простая модель рабочего листа с несколькими элементами управления. На рабочем листе находятся переключатели, кнопки, поле, а также поле со списком.

НА КОМПАКТ-ДИСКЕ

Эта рабочая книга, в которой также содержится несколько простых макросов, представлена на прилагаемом к книге компакт-диске.

Возможно, самым распространенным элементом управления является `CommandButton`. Сами по себе кнопки не выполняют никаких функций, но каждой из них вы можете назначить макрос.

Использование элементов управления непосредственно на рабочем листе отменяет необходимость в создании пользовательских диалоговых окон. Часто бывает так, что вставка в рабочий лист нескольких элементов управления ActiveX значительно упрощает работу с электронной таблицей. В результате пользователь сможет выбирать то, что ему нужно, работая со знакомыми элементами управления, а не вводя значения в ячейки.

Элементы управления ActiveX отображаются на панели инструментов Элементы управления. Кроме того, на рабочем листе пока еще можно использовать элементы управления, совместимые с Excel 5/95. Они не относятся к категории элементов управления ActiveX и находятся на панели инструментов Формы. Детально эти элементы управления в книге не описаны. Однако ниже приведены сравнительные данные по этим двум типам элементов управления (табл. 6.1).

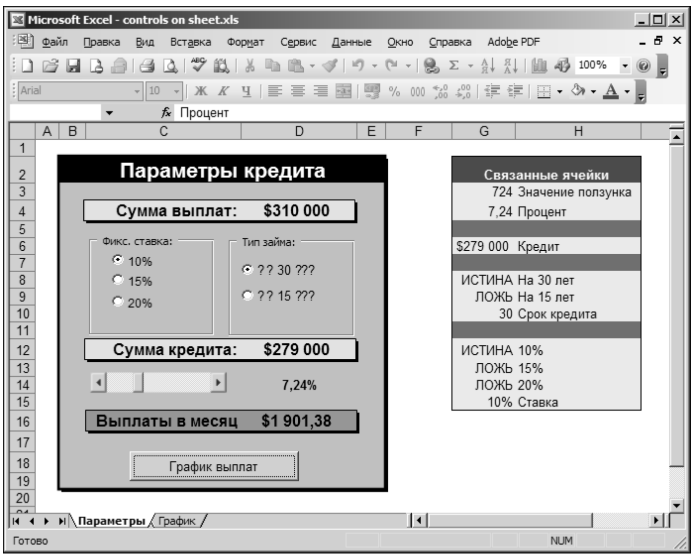


Рис. 6.1. Если элементы управления, предназначенные для пользовательских форм, вставить непосредственно на рабочий лист, то это может существенно облегчить его использование

Таблица 6.1. Сравнение элементов управления программы Excel с элементами управления ActiveX

	Элементы управления ActiveX	Элементы управления Excel
Версии Excel	97, 2000, 2002	5, 95, 97, 2000, 2002
Панель инструментов	Элементы управления	Формы
Хранение кода макросов	В модуле кода	В любом стандартном VBA-модуле листа
Имя макроса	Соответствует имени элемента управления (например, CommandButton1_Click)	Любое указанное вами имя
Соответствует ...	Элементам управления пользовательских форм	Элементам управления диалоговых листов
Настройка	В широком объеме, с помощью панели Properties	Минимальная
Реакция на события	Да	Только на события `Click` (Щелчок) или `Change` (Изменение)

Настройка меню

В приложениях электронных таблиц управлять пользовательским интерфейсом можно самыми разными способами. Речь идет об изменении стандартных меню Excel или создании собственных систем меню. Вместо того чтобы создавать кнопки для выполнения макросов, можете добавить одно или несколько меню (а также элементов меню) и уже с их помощью запустить созданные ранее макросы. Преимущество пользовательских меню заключается в том, что строка меню всегда отображается на экране, а кнопка, размещенная на рабочем листе, при прокрутке может быстро исчезнуть из поля зрения.

Начиная с Excel 97, компания Microsoft реализует технологию управления меню, которая отличается исключительной оригинальностью. Как вы узнаете из главы 22, строка меню является замаскированной панелью инструментов. На рис. 6.2 показан образец меню, добавленного в среду программы Excel. Это меню создано с помощью надстройки Power Utility Pak. Каждый элемент меню предназначен для запуска какого-либо макроса.

Существует два способа настройки меню Excel. Изменения в меню можно внести с помощью кода VBA или посредством редактирования самих меню (для этого воспользуйтесь командой Вид⇨Панели инструментов⇨Настройка).

Изменения в меню рекомендуется проводить с помощью команд VBA (см. главу 22). В коде VBA вы можете управлять меню самым произвольным образом: отключать команду меню или отменять установку опции (переключателя) меню.

Изменения в меню, которые выполняются с помощью команды Вид⇨Панели инструментов⇨Настройка (рис. 6.3), "постоянные". Другими словами, если подобным образом изменить состав меню (например, удалить элемент меню), то результат этой операции останется в силе даже после перезапуска Excel.

Начиная с Excel 97, вы не найдете в программе редактор меню, который был представлен в Excel 5. Когда рабочая книга загружается в среде Excel 97 или более поздней версии, то меню, созданные с помощью редактора меню, все равно продолжают использоваться. Впрочем, видоизменить или удалить меню, созданные посредством этого редактора, все же можно, однако только двумя способами: используя Excel 5 или установив специальную утилиту, предназначенную для выполнения необходимых действий.

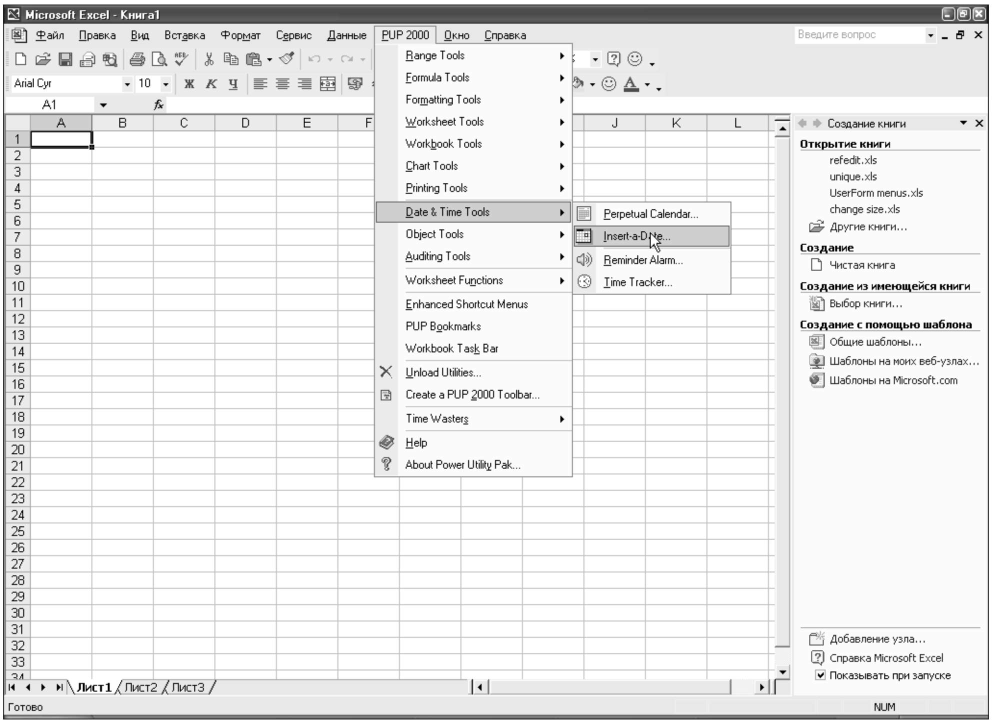


Рис. 6.2. Данное меню создано с помощью надстройки

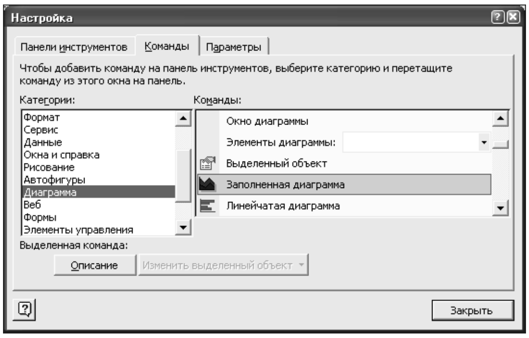


Рис. 6.3. В диалоговом окне Настройка *можно вручную изменить строку меню Excel*

Вы вскоре убедитесь в том, что настраивать можно любые меню, отображаемые программой Excel, даже контекстные, которые появляются при щелчке на объекте правой кнопкой мыши. Впрочем, настроить контекстные меню можно только с помощью VBA (посредством элементов графического интерфейса этого делать нельзя). На рис. 6.4 показано пользовательское контекстное меню, которое появляется при щелчке правой кнопкой мыши на ячейке или диапазоне ячеек. Обратите внимание, что в этом меню находятся только новые команды, которые по умолчанию не используются.

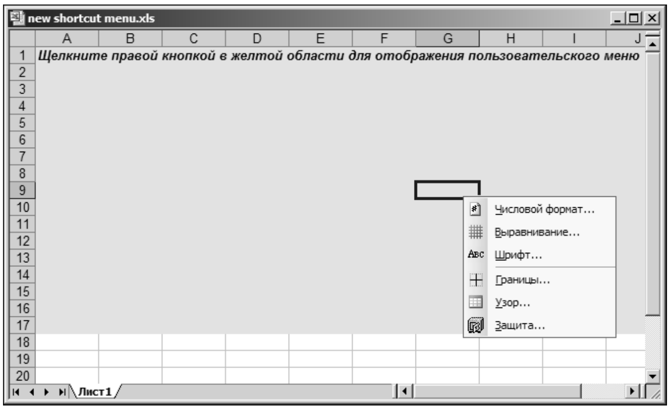


Рис. 6.4. Пример пользовательского контекстного меню



Подробно о пользовательских меню рассказано в главе 23.

Настройка панелей инструментов

Панели инструментов широко распространены в приложениях Windows, и в программу Excel встроено огромное их количество. Обычно кнопки панелей инструментов помогают пользователям быстрее выполнять наиболее распространенные команды меню. Так как для щелчка на кнопке, которая расположена на панели инструментов, обязательно требуется мышь, то такая кнопка — далеко *не единственное* средство выполнения конкретной операции. Однако в Excel большинство самых распространенных операций в электронных таблицах выполняется без использования меню, т.е. для их вызова вполне хватает панелей инструментов.

Вы можете создать пользовательскую панель инструментов, на которой будут содержаться кнопки только тех команд, которые, как вам кажется, необходимо сделать доступными для пользователей. Если с этими кнопкам связаны макросы, то пользовательская панель инструментов станет эквивалентной группе кнопок, расположенных на рабочем листе. Однако применять панель инструментов удобнее, поскольку она всегда отображается на экране, и ее перемещение в другое место не повлияет на основной документ. Что же касается кнопок, размещенных на рабочем листе, то они жестко прикреплены к определенному месту, и их можно убрать из поля зрения в результате прокрутки документа.



Начиная с Excel 97, на панель инструментов позволяется также добавлять меню.

Вы можете таким образом настроить свое приложение, что при его загрузке на экране будет появляться соответствующая панель инструментов. Присоедините эту панель к рабочей книге, используя кнопку Вложить, которая находится на вкладке Панели инструментов диалогового окна Настройка (рис. 6.5). Тогда вместе с приложением — рабочей книгой вы сможете хранить специально подготовленные панели инструментов, и эти панели можно будет передавать другим пользователям приложения.

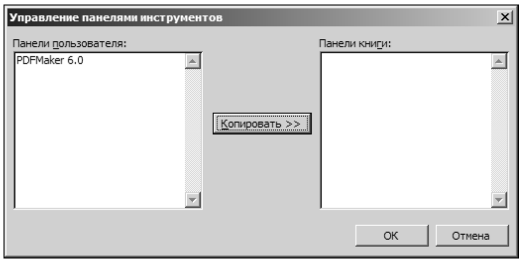


Рис. 6.5. Пользовательскую панель инструментов вы вправе присоединить к рабочему листу с помощью диалогового окна Управление панелями инструментов

Подробно о панелях инструментов рассказывается в главе 22.

Создание комбинаций клавиш

Наконец, мы приступаем к рассмотрению последнего инструмента пользовательского интерфейса, которым вы располагаете. Этим инструментом являются пользовательские комбинации клавиш. Excel позволяет назначать макросу комбинацию клавиш, в состав которой входит <Ctrl> (или <Shift+Ctrl>). Когда пользователь нажимает комбинацию клавиш, макрос запускается и выполняет все определенные в нем действия. Очевидно, вы должны объяснить пользователю, какие комбинации клавиш добавлены в приложение, и какие действия они выполняют. Если существенную роль в управлении приложением играет скорость выполнения операций, то комбинации клавиш использовать эффективнее, поскольку на их нажатие обычно уходит меньше времени, чем на выбор команд меню, применение панели инструментов или опций диалоговых окон.

Впрочем, остерегайтесь назначать ту комбинацию клавиш, которая уже используется для выполнения других операций. Например, <Ctrl+S> применяется для сохранения текущей рабочей книги. И если вы присвоите эту комбинацию одному из макросов, то не сможете с ее помощью сохранить файлы. Другими словами, комбинация клавиш, присвоенная макросу, имеет приоритет над теми комбинациями, которые встроены в Excel. Комбинации клавиш чувствительны ко всем модифицирующим клавишам, поэтому вы можете использовать такие варианты, как, например, <Ctrl+Shift+S>.

Усилия по разработке приложения

Когда вы определите потребности пользователя, подберете тип проекта, который должен их удовлетворить, и решите, какие компоненты необходимо применить в пользовательском интерфейсе, настанет время завершить подготовительные операции и приступить к созданию самого приложения. Этот этап, конечно же, занимает значительную часть времени, которое тратится на реализацию проекта.

Каким образом вы будете создавать приложение, зависит от вашего персонального стиля работы и от природы самого приложения. Если ваше приложение не является простой рабочей книгой, содержащей шаблоны бланков для заполнения, то в нем, скорее всего, будут использоваться макросы. Написание макросов занимает много времени и сил. Не думайте, что создавать макросы невероятно сложно — трудно создавать *хорошие* макросы.

Работа с конечным пользователем

В настоящем разделе речь пойдет о весьма важных проблемах разработки, которые дают о себе знать, когда приложение становится работоспособным и приближается время размещать готовый проект в компьютерах конечных пользователей.

Тестирование приложения

Сколько раз при использовании коммерческого приложения оно "отказывает" в самый неподходящий момент? Скорее всего, проблема вызвана недостаточно качественным тестированием, в результате которого не обнаружены все ошибки. Конечно, ошибки имеются у всех коммерческих программ, просто в самых лучших программах ошибки заметить труднее. Вы часто будете сталкиваться с ситуацией, когда необходимо обойти ошибки, не отслеженные в самой программе Excel, иначе ваше приложение, как положено, работать так и не будет.

Создав приложение, вы должны его протестировать. Это один из самых важных этапов. Нередко бывает так, что на тестирование и отладку приложения уходит столько же времени, сколько тратится на создание его исходного варианта. В конце концов, пишете вы процедуру VBA или создаете формулы рабочего листа, вам все равно захочется убедиться, что приложение работает именно так, как и предполагалось.

Ошибки? В программе Excel?

Вы, возможно, думаете, что такой продукт, как Excel, используемый миллионами людей по всему миру, не должен содержать (по крайней мере грубых) ошибок. Спешим вас обрадовать — это не так. Программа Excel является настолько сложной, что вполне естественно ожидать от нее неадекватных действий. И, *конечно же*, подобные действия вызываются разного рода ошибками.

Создать такой программный продукт, как Excel, — задача не из легких даже для компании Microsoft с ее явно неограниченными ресурсами. При выпуске программы нельзя избежать компромиссов. Общеизвестно, что крупные производители программных продуктов выпускают свои приложения, прекрасно осознавая, что те содержат ошибки. Большинство этих ошибок настолько незначительны, что на них вообще можно не обращать внимания. Конечно, компании — производители программ могли бы повременить с выпуском своих продуктов на пару месяцев и исправить большинство имеющихся ошибок. Но программы, как и другие товары, являются "скоропортящимися" продуктами. Прибыли от задержанного по выпуску продукта часто меньше израсходованных на него средств. И хотя в программе Excel имеются ошибки, большинство ее пользователей никогда с этими ошибками не столкнется.

В данной книге указаны важные проблемы использования Excel. Кроме них, вы, конечно же, найдете и другие ошибки. Некоторые проблемы характерны только для конкретной версии Excel и в определенной аппаратно-программной конфигурации. Это наихудшие из ошибок, потому что их распознать весьма нелегко.

Так как же быть разработчику приложений? Именно так, как призывает доктрина *обхода ошибок*. Если то, что вы пытаетесь делать, не работает, хотя все говорит о том, что *должно* работать, то самое время перейти к Плану Б. Обидно? Конечно же. Потеря времени? Гарантированно. Но без всего этого нельзя представить судьбу разработчика.

Склонность к ошибкам имеют не только стандартные компилированные приложения. То же самое можно сказать и о приложениях электронных таблиц. *Ошибка* обычно определяется следующим образом: (1) это нечто такое, что при выполнении программы (или приложения) происходит, но происходить не должно; (2) это нечто такое, что не происходит, но происходить как раз обязано.

Оба вида ошибок одинаково отвратительны. Поэтому значительную часть времени, которое уйдет на разработку, вы должны заранее выделить на тестирование приложения во всех возможных условиях и на исправление любых встретившихся проблем. К сожалению, иногда не только вы виноваты в возникающих проблемах. Свою "лепту" вносит и сама программа Excel (см. врезку "Ошибки? В программе Excel?").

Вероятно, не требуется напоминать об обязательности тщательного тестирования любых электронных таблиц, которые вы разрабатываете для других. Учитывая возможную аудиторию вашего приложения, вы можете сделать его *отказоустойчивым*. Другими словами, попробуйте прогнозировать все возможные ошибки и недочеты, которые потенциально могут допускаться пользователями, и приложите усилия, чтобы их избежать (или хотя бы красиво скрыть). Это не только поможет конечному пользователю, но и позволит не запятнать вашей репутации.

Конечно, не в ваших силах прогнозировать все возможности, однако создаваемые макросы должны обрабатывать основные виды ошибок. Например, пользователь вместо числового значения ввел текстовое, или он пытается запустить ваш макрос, когда рабочая книга еще не открыта. Он может скрыть диалоговое окно, так ничего и не настроив, или нажать комбинацию клавиш <Ctrl+F6> и перейти к следующему окну. Что делать вам в этих ситуациях? Когда вы приобретете достаточный опыт, то такого рода вопросы станут для вас обязательными при тестировании приложений, и вы будете отвечать на них без лишних раздумий.

А как же бета-тестирование?

Производители программ обычно предусматривают для своих новых продуктов процедуры тщательного тестирования. После широкого внутреннего тестирования выпуску продукта на рынок обычно предшествует тестирование его группой заинтересованных пользователей. Такая операция приобрела название *бета-тестирования*. На этом этапе часто обнаруживаются новые проблемы, обычно исправляемые ко времени выпуска конечного продукта.

Если вы разрабатываете приложение в Excel, которым будет пользоваться гораздо больше, чем несколько человек, то вам, возможно, захочется провести его бета-тестирование. Оно позволит запустить созданное приложение в предназначенном для него программном окружении на (как правило) разном оборудовании и именно теми пользователями, для которых оно предназначено.

Период бета-тестирования начинается после завершения самостоятельного тестирования приложения и перед распространением полученного продукта среди пользователей. Определите группу пользователей, которые будут вам помогать тестировать продукт. Лучшие результаты бета-тестирования достигаются тогда, когда бета-тестеры получат абсолютно все, что входит в пакет поставки приложения: программную документацию, программу установки, компьютерную справочную систему и т.д. Узнать результаты бета-тестирования можно по-разному, в том числе путем личного общения, анкетирования или по телефону.

Перед тем, как вы отважитесь на распространение своего продукта среди пользователей, вы наверняка столкнетесь с проблемами, для решения которых потребуется внести дополнительные исправления или провести корректировку. Конечно, на бета-тестирование требуется дополнительное время, и не все проекты (и разработчики) могут позволить себе такую роскошь.

Как сделать приложение отказоустойчивым

Нарушить электронную таблицу можно очень даже легко. Часто бывает так, что удаление одной важной формулы или ключевого значения вызывает ошибки во всей таблице — или даже в других зависимых таблицах. Более того, если поврежденную таблицу сохранить, то на диске она заменит корректную копию. И если резервное копирование приложения вами не выполнялось, то его пользователь будет *очень* расстроен, в чем, скорее всего, обвинит именно *вас*.

Теперь ясно, зачем производится дополнительная защита приложения, прежде чем пользователи (особенно начинающие) приступят к его применению для решения конкретных задач. В Excel для защиты рабочих листов и их частей можно использовать несколько способов.

- *Заблокировать определенные ячейки*, чтобы предотвратить их изменение. (Задается с помощью вкладки Защита диалогового окна Формат ячеек.) Блокировка вступает в силу только тогда, когда с помощью команды Сервис⇨Защита⇨ Защитить лист документ защищается паролем.
- *Добавить защиту всей рабочей книги* — ее структуры, расположения и размеров окна или одновременно всех перечисленных выше параметров. Это можно сделать с помощью команды Сервис⇨Защита⇨Защитить книгу.
- *Скрыть формулы*, расположенные в ячейках, чтобы их не смогли увидеть другие пользователи. (Устанавливается с помощью вкладки Защита диалогового окна Формат ячеек.) Формула будет скрыта только тогда, когда с помощью команды Сервис⇨Защита⇨Защитить лист документ будет защищен паролем.

Защита данных паролем в Excel

Известно, что компания Microsoft никогда не рекламировала Excel в качестве полностью защищенной программы. И тому есть определенная причина — вывести из строя систему паролей Excel на самом деле довольно легко. Существует несколько коммерческих программ, с помощью которых можно взламывать пароли. Впрочем, у программы Excel 2003 защита более мощная, чем у предыдущих версий. И что, этого достаточно? Не стоит считать защиту Excel абсолютно надежной. Конечно, при вторжении случайного пользователя она устоит. Но если кому-то *действительно* захочется взломать пароль, то он, скорее всего, добьется результата.

- *Заблокировать объекты рабочего листа.* (Устанавливается с помощью вкладки Защита диалогового окна Формат объекта.) Эта блокировка вступает в силу только тогда, когда с помощью команды Сервис⇨Защита⇨Защитить лист документ защищается паролем.
- *Скрыть строки* (команда Формат⇨Строка⇨Скрыть), столбцы (команда Формат⇨Столбец⇨Скрыть), листы (команда Формат⇨Лист⇨Скрыть) и документы (Окно⇨Скрыть). В результате этих операций лист не будет выглядеть загроможденным, однако он частично защищен от любопытных глаз.
- *Превратить рабочую книгу Excel в документ только для чтения* (и защищенную паролем), чтобы ее файл нельзя было изменить. Этот режим устанавливается в диалоговом окне Параметры сохранения. Запустите его из диалогового окна Сохранение документа с помощью команды Сервис⇨Общие параметры.
- *Добавить пароль*, чтобы неавторизованный пользователь не мог открыть ваш файл. Такой режим можно установить в диалоговом окне Параметры сохранения. Чтобы его отобразить, выберите в диалоговом окне Сохранение документа команду Сервис⇨Общие параметры.
- *Использовать защищенные паролем надстройки*, чтобы пользователь не мог изменить рабочие листы самой надстройки.

Начиная с Excel 2002, вам предоставляются возможности, которые связаны с защитой листов. Выполните команду Сервис⇨Защита⇨Защитить лист и внимательно рассмотрите диалоговое окно Защита листа. С его помощью можно точно указать, какие действия необходимо выполнить для защиты листа. Кроме того, в диалоговом окне Параметры сохранения расположена кнопка Дополнительно. Щелкнув на ней, вы сможете установить тип шифрования при защите рабочей книги.

Привлекательное и наглядное приложение

Если вам приходилось использовать несколько программных пакетов, то вы, несомненно, видели плохо разработанные пользовательские интерфейсы, трудные в применении программы и просто отвратительные диалоговые окна. Разрабатывая электронные таблицы для других пользователей, не забывайте уделить особое внимание внешнему виду своего приложения.

У конечных пользователей первая реакция на приложение вызвана исключительно внешнем видом программы. Это в полной мере относится и к приложениям, которые вы разрабатываете с помощью Excel. У каждого, впрочем, свое представление о красоте. И если вы вышли на серьезный уровень профессиональной деятельности, то подумайте над тем, чтобы к разработке приложения привлечь человека с хорошим вкусом.

Вы должны дополнительно уделить внимание вопросам дизайна и эстетики — и пользователи обязательно оценят ваше умение разрабатывать не только удобный, но и красивый интерфейс. Само же приложение будет выглядеть утонченно и более профессионально. Хороший внешний вид приложения говорит о том, что разработчик настолько заботился о своем продукте, что не пожалел времени и усилий на создание всесторонне качественного интерфейса.

- Добивайтесь единообразия. Разрабатывая, например, диалоговые окна, старайтесь по возможности следовать внешнему виду диалоговых окон Excel. Соблюдайте единообразие в формате, шрифтах, размере текста и цветах.
- Распространенной ошибкой разработчиков является то, что они пытаются вывести на экран или в диалоговом окне как можно больше информации. Добиваясь этого, следуйте эмпирическому правилу, согласно которому информацию необходимо выдавать порциями — не более одного-двух блоков за раз.
- Если для получения информации от пользователя вы используете диалоговое окно, то подумайте над тем, чтобы разбить его на несколько окон, каждое из которых будет менее загроможденным. Если вы все же решили использовать сложное диалоговое окно, то чтобы его разбить, можете воспользоваться элементом управления `MultiPage` (Множество вкладок), который помогает создать знакомое вам диалоговое окно с несколькими вкладками.
- Цвета используйте осторожно, потому что с ними легко переусердствовать.
- Уделите особое внимание числовым форматам, начертанию, размерам шрифтов, а также границам.

Эстетика — понятие довольно субъективное, но если вы сомневаетесь в своих способностях, то постарайтесь добиться как минимум простоты и ясности интерфейса приложения.

Создание пользовательской справочной системы

Что же касается пользовательской документации, то в ней также используется преимущественно два варианта: документация в бумажном и электронном виде. Электронная справочная система — это стандартный компонент Windows-приложений. Ваши Excel-приложения не исключения — они могут иметь электронную справочную систему и даже ее контекстную разновидность. На разработку электронной справочной системы требуется много времени и усилий, но в большом проекте они, конечно же, с лихвой окупятся.

Кроме указанных выше рекомендаций, вы должны уделить особое внимание технической поддержке приложения в будущем. Иными словами, кому будут звонить пользователи при возникновении проблем с его использованием? Если вы не готовы

отвечать на такие вопросы, то найдите того, кто будет этим заниматься. В большинстве случаев вам захочется отвечать только на сложные “технологические” вопросы, касающиеся исправления ошибок программного кода.

В главе 24 рассмотрены принципы предоставления справочных сведений в пользовательских приложениях.

Документирование усилий, потраченных на разработку

Собрать приложение электронных таблиц в единое целое — это одна задача, а сделать приложение понятным для других людей — совсем другая. Как и в традиционном программировании, в данном случае важно тщательно документировать свою работу. Созданная вами документация пригодится тогда, когда вам потребуется вернуться к доработке приложения (до этого, конечно же, у вас когда-нибудь дойдут руки). Она также понадобится тому, кому вы в дальнейшем передадите свое приложение.

При документировании проекта вам, скорее всего, придется учесть следующий фактор. Если вас пригласили для разработки приложения Excel, то вам, видимо, не захочется, тщательно документируя все и вся, делиться, таким образом, всеми секретами, заработанными тяжелым трудом. В этом случае необходимо подготовить два варианта документации: подробный и сокращенный.

Как же документировать приложение электронных таблиц? Информацию можно хранить в рабочей книге или другом файле. Если хотите, можете представить документацию в виде бумажного документа. Возможно, самый легкий способ — это использовать отдельный рабочий лист, чтобы хранить в нем ваши комментарии и основную информацию о проекте. Что касается кода VBA, то вы можете использовать в нем комментарии (ведь код, перед которым введен апостроф, все равно игнорируется). Элегантный фрагмент кода VBA сегодня вам может казаться предельно очевидным, но, изучив этот фрагмент через пару месяцев, вы полностью запутаетесь в приводимых ранее рассуждениях.

Распространение приложения среди пользователей

Вы завершили проект и готовы передать его конечному пользователю. Каким образом вы это сделаете? Можете выбрать один из многих доступных способов распространения программных продуктов. Выбор способа зависит от многих факторов.

Создайте диск с приложением и внесите в него простые инструкции по использованию. Вам, конечно, захочется самому установить созданное приложение — но такое не всегда возможно. Существует еще один вариант — создать официальную программу установки. Эта программа предназначена для автоматического выполнения всех необходимых операций по внедрению приложения в компьютер пользователя. Такую программу вы можете написать на традиционном языке программирования, купить общий ее код или написать свою собственную на VBA.

В Excel 2000 и более поздних версиях используется технология Microsoft Authenticode, благодаря которой разработчики могут скреплять свои приложения “цифровой подписью”. Эта технология должна помочь конечным пользователям распознавать автора приложения, гарантировать, что проект не изменен сторонними пользователями, и препятствовать распространению макровирусов и другого потенциально разрушительного кода. Чтобы защитить проект цифровой подписью, вначале следует обратиться в специальную организацию по сертификации и получить соответствующий сертификат. (Существует и другая возможность — самостоятельно подписать свой проект, создав уникальный цифровой сертификат). Дополнительная информация на эту тему приведена в электронной справочной системе и на Web-узле компании Microsoft.

Почему нет исполняемой версии Excel?

Когда вы размещаете свое приложение в компьютере конечных пользователей, то должны быть уверены, что каждый из них имеет лицензионную копию необходимой версии Excel. Передача копии Excel вместе с приложением является незаконной. У вас, возможно, возникнет вопрос: почему же Microsoft не создала исполняемую версию Excel? Исполняемая версия — это программа, которая может загружать файлы, но не создавать их. Имея такую версию, пользователи не нуждались бы в копии Excel для запуска вашего приложения (как часто бывает с приложениями баз данных).

Трудно найти ясное и убедительное объяснение, почему Microsoft не распространяет исполняемой версии Excel и почему подобной маркетинговой концепции придерживаются другие производители процессоров электронных таблиц. Наиболее вероятная причина заключается в следующем: производители боятся, что из-за выпуска исполняемой версии уменьшатся объемы продаж полных версий программ. Или, возможно, разработка исполняемой версии требует дополнительного перепрограммирования имеющегося кода, что никогда не окупится на современном рынке программных продуктов.

Извещение на тему… Компания Microsoft настойчиво предлагает *программу просмотра* файлов Excel. Эта утилита предоставляет возможность просматривать файлы Excel без установки копии Excel. Правда, макросы в программе просмотра не выполняются. Копию бесплатной программы просмотра можно получить на Web-узле компании Microsoft (`http://officeupdate.microsoft.com`).

Обновление приложения

Распространив приложение среди пользователей, вы завершили процесс его разработки, не правда ли? И теперь можно отдохнуть, наслаждаясь жизнью и пытаясь забыть о проблемах, встретившихся вам (и решенных вами же) при разработке приложения. Конечно, в редких случаях приложение действительно можно считать завершенным. Однако чаще случается так, что пользователи приложения не будут полностью им довольны. Всем *первоначальным* спецификациям оно будет соответствовать, но жизнь ведь не стоит на месте. Когда пользователь встречает правильно работающее приложение, то начинает думать о функциях, которыми можно его пополнить. Поэтому он попросит об их включении в приложение. Да, мы говорим об *обновлениях*.

Рано и поздно вам потребуется обновить или переделать приложение. Вот тут вы и оцените, насколько хорошо спроектировали его вначале и полностью ли задокументировали все поддерживаемые операции. Если нет, то... все мы учимся на собственном опыте.

Другие вопросы разработки приложений

При разработке приложения вам следует помнить также и о некоторых других вопросах — особенно тогда, когда вы не знаете точно, кто же будет использовать это приложение. Если вы разрабатываете приложение, которое планируется применять в широком кругу пользователей (например, создается условно-бесплатное приложение), то сложно предугадать, каким образом его будут использовать, в какой системе и вместе с какими программами оно будет запускаться.

Версия Excel, установленная у пользователя

С каждой новой версией Excel вопрос совместимости становится все более актуальным. На время написания этой книги программа Excel 2003 уже продавалась, но во многих больших корпорациях по-прежнему продолжают использовать Excel 95, Excel 97 и даже более ранние версии программы.

К сожалению, нет никакой гарантии того, что приложение, разработанное, скажем, для Excel 95, будет прекрасно выполняться в последующих версиях Excel. Если необходимо, чтобы ваше приложение работало в Excel 95, Excel 97, Excel 2000 и Excel 2002, то вам потребуется работать с самой старой версией программы (Excel 95), а затем протестировать свой продукт во всех остальных версиях.

Дело еще более усложнится, если учитывать "подверсии" программы Excel. Для исправления тех или иных проблем Microsoft выпускает сервисные пакеты. Например, пользователи могут работать с версиями Excel 2000, Excel 2000 с SR-1 или Excel 2000 с SR-2.

Вопросы совместимости рассматриваются в главе 26.

Трудности, касающиеся поддержки языка

Считайте, что вам повезло, если все конечные пользователи вашего приложения имеют англоязычную версию Excel. Дело в том, что версии этой программы на других языках не всегда полностью совместимы. Поэтому вам придется проводить дополнительное тестирование поддержки приложения в многоязычной среде.

Проблемы использования иностранных языков кратко рассматриваются в главе 26.

Производительность системы

Вероятно, вы — достаточно опытный пользователь компьютера — стараетесь постоянно обновлять его оборудование. Другими словами, у вас достаточно производительная система, которая лучше, чем система среднестатистического пользователя. В отдельных случаях вам точно известно, какое аппаратное обеспечение используется конечными пользователями ваших приложений. Если это так, то крайне важно, чтобы вы протестировали приложение в подобной аппаратной среде. Может получиться так, что процедура, которая выполняется в вашей системе почти мгновенно, в другой системе потребует нескольких секунд. А ведь несколько секунд "простоя", потраченные на выполнение простой операции, могут не удовлетворить искушенного современными технологиями пользователя.

Приобретая все больший опыт работы с VBA, вы обнаружите, что способы выполнения работы и *быстрого* выполнения работы существенно отличаются. Выработайте привычку вводить программный код "на скорость". В этом, конечно же, вам помогут другие главы этой книги.

Видеорежимы

Большинство пользователей Windows применяют один из трех стандартных видеорежимов: 640×480 (т.е. стандартный режим VGA), 800×600 и 1024×768. Если вы разрабатываете приложение для режима, отличного от VGA, то, запущенное в VGA-системе, оно будет выглядеть ужасно.

Ваше приложение может отображаться на единственном экране, и это порой вызывает определенные трудности. Например, если вы разрабатываете диалоговое окно ввода данных, используя режим 800×600, то пользователи со стандартным VGA-монитором не всегда без прокрутки смогут увидеть все окно. Кроме того, важно понимать, что восстановленная (т.е. неразвернутая и несвернутая) рабочая книга ото-

бражается с предыдущими размерами окна и в предыдущей позиции. Случается и так, что окно, сохраненное при отображении с высоким разрешением, полностью смещается за пределы экрана, когда открывается в VGA-системе.

Несмотря на то, что опытные пользователи обычно применяют в Windows более высокие разрешения, в некоторых случаях использование режима VGA просто неизбежно. Например, во встроенных мониторах некоторых старых портативных компьютеров поддерживается только режим VGA. Не существует способа, который позволяет автоматически масштабировать на экране элементы приложения, чтобы при любом разрешении они выглядели одинаково. И если вы не уверены в том, какие разрешения экрана применяются пользователями приложения, то важно, чтобы вы его спроектировали, основываясь на “наименьшем из возможного” — режиме VGA.

Как вы узнаете позже (глава 10), разрешение экрана в компьютере пользователя можно определять с помощью кода VBA. В некоторых случаях вам, возможно, захочется программно регулировать появляющиеся на экране элементы приложения в соответствии с используемым разрешением.

Часть III

Visual Basic for Applications

В этой части...

Глава 7

Введение в Visual Basic for Applications

В ЭТОЙ ГЛАВЕ...

Программирование в Excel, в основном, сводится к управлению объектами. Эта задача выполняется с помощью инструкций, которые вводятся на языке, понятном Excel.

- Введение в VBA — язык программирования, встроенный в Excel
- Отличие VBA от других языков программирования
- Использование редактора Visual Basic
- Окна редактора Visual Basic и настройки среды программирования
- Запись макросов Excel
- Обзор объектов, коллекций, свойств и методов
- Пример использования объекта `Comment`
- Дополнительная информация и примеры работы с объектами `Range`
- Способы получения информации об объектах, свойствах и методах Excel

Настоящая глава ознакомит вас с языком VBA, а также с объектами, включенными в Excel.

История языка BASIC

Многие опытные программисты не воспринимают идею программирования на BASIC всерьез. Само название (аббревиатура от **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode — универсальный символический язык инструкций для начинающих) предполагает, что это не профессиональный язык. Действительно, BASIC был разработан в начале 1960-х годов и задумывался как наглядное средство преподавания методов программирования студентам колледжей. BASIC довольно быстро приобрел большую популярность, и сейчас он поддерживается многими типами компьютеров.

С годами BASIC развивался и улучшался. Например, во многих ранних версиях он был *интерпретируемым* языком. Каждая строка перед выполнением интерпретировалась, чем и была обусловлена медленная скорость обработки кода. В большинстве современных вариантов языка BASIC программа компилируется. В результате программа выполняется значительно быстрее, ее перемещаемость улучшилась.

BASIC стал намного популярнее в 1991 году, когда компания Microsoft выпустила Visual Basic для Windows. Этот продукт облегчил массовую разработку самостоятельных приложений для Windows. Visual Basic имеет немного общего с ранними версиями BASIC, но последний представляет собой основу, на которой построен VBA.

Обзор VBA

Excel 5 — это первое приложение на рынке программного обеспечения, в котором появился Visual Basic for Applications (VBA). VBA считается стандартным языком написания сценариев для приложений Microsoft, и в настоящее время он входит в состав всех приложений Office 2003 и даже приложений других компаний. Следовательно, овладев VBA для Excel, вы сможете сразу перейти к созданию макросов для других программных продуктов Microsoft (равно, как и приложений других компаний). Более того, вы сможете создавать полноценные программные продукты, одновременно использующие функции самых разных приложений.

Объектные модели

Секрет использования VBA заключается в правильном понимании *объектной модели* в каждом отдельном приложении. Следует отметить, VBA всего лишь управляет объектами, а каждый программный продукт (Excel, Word, Access, PowerPoint и т.п.) имеет свою объектную модель. Приложением можно управлять программным образом только с помощью объектов, которые *представлены* в этом приложении.

Например, в объектной модели Excel представлено несколько мощных объектов анализа данных, например, рабочие листы, диаграммы, сводные таблицы, сценарии, а также многочисленные математические, финансовые, инженерные и общие функции. С помощью VBA вы можете работать с этими объектами и разрабатывать автоматизированные процедуры. В процессе изучения VBA в Excel вы сможете получить полное представление об объектной модели. Сначала она покажется вам невероятно сложной. Однако со временем разрозненные фрагменты будут собраны в единую картину, и вы поймете, что овладели очень сложной темой!

Сравнение VBA и XLM

До появления Excel 5 разработчиками использовался мощный (но сложный для понимания) язык макросов под названием XLM. Более поздние версии Excel все еще выполняют макросы XLM, но, начиная с Excel 97, пользователи не имеют возможности записывать макросы на языке XLM. Как разработчик вы должны знать о существовании XLM (на случай, если столкнетесь с макросами, написанными на этом языке), но для собственных разработок используйте исключительно VBA.

На рис. 7.1 и 7.2 показана простая процедура, написанная на XLM и на VBA. Рассматриваемый макрос управляет выделенными ячейками. Он изменяет цвет фона ячеек и добавляет к ним границы. Вероятно, вы уже сейчас видите, что код VBA понимать намного легче. Кроме того (и что более важно), программу VBA проще изменить в случае необходимости.

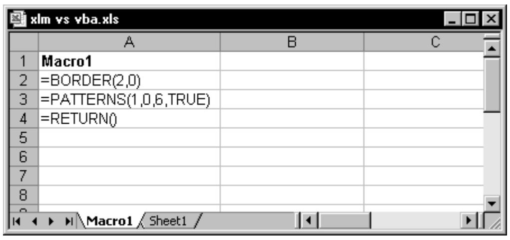


Рис. 7.1. Простой макрос, написанный на языке XLM Excel и сохраненный на листе макросов

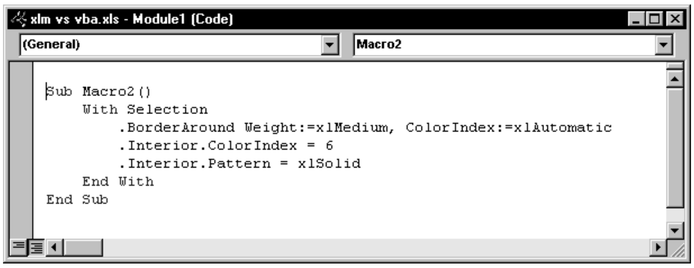


Рис. 7.2. Простой макрос, написанный на языке VBA Excel и сохраненный в модуле VBA

Основы VBA

Прежде чем непосредственно перейти к рассмотрению методов программирования, ознакомьтесь с материалом этого раздела, представляющим краткий обзор следующих разделов главы. Ниже указаны темы, которые будут рассмотрены в оставшейся части главы.

Остановимся на предназначении VBA.

- Действия в VBA осуществляются в результате выполнения кода VBA.
- Вы создаете (или записываете) программу VBA, которая сохраняется в модуле VBA.

  Модуль VBA состоит из процедур.
- Процедура, по существу, представляет собой элемент компьютерной программы, выполняющей определенное действие.

  Ниже приведен пример простой процедуры под названием `Test`: она вычисляет сумму, а затем отображает результат в окне сообщений.

```
Sub Test()
  Sum = 1 + 1
  MsgBox "Ответ: " & Sum
End Sub
```

- Кроме процедур `Sub` в модуле VBA может использоваться второй тип процедур — функции.

  Функция возвращает одно значение (или массив). Функция может быть вызвана из другой процедуры VBA или использоваться в формуле рабочего листа. Далее приведен пример функции с названием `AddTwo`.

```
Function AddTwo(arg1, arg2)
  AddTwo = arg1 + arg2
End Function
```

- VBA управляет объектами, которые представлены запускающим приложением (в данном случае Excel).

  Excel позволяет управлять более, чем ста классами объектов, включая рабочую книгу, рабочий лист, диапазон ячеек рабочего листа, диаграмму и нарисованный прямоугольник. В вашем распоряжении находятся и другие объекты, с которыми можно работать в VBA.

  Классы объектов организованы в иерархическую структуру.

  Объекты могут выступать контейнерами других объектов. Например, Excel — это объект под названием `Application`, он содержит другие объекты, например,

`Workbook` (Рабочая книга). Объект `Workbook` может состоять из других объектов, например, `Worksheet` (Рабочий лист) и `Chart` (Диаграмма). Объект `Worksheet` также содержит объекты, например, `Range` (Диапазон), `PivotTable` (Сводная таблица) и т.д. Организацию таких объектов называют *объектной моделью* Excel.

- Одинаковые объекты формируют *коллекцию*.

  Например, коллекция `Worksheets` состоит из всех рабочих листов конкретной рабочей книги, а коллекция `CommandBars` — из всех объектов `CommandBar`. Коллекции — это объекты в себе.

- При ссылке на объект, вложенный в другой объект, положение в иерархической структуре объектной модели задается с помощью точки-разделителя.

  Например, на рабочую книгу с названием `Книга1.xls` можно сослаться следующим образом.

  ```
  Application.Workbooks("Книга1.xls")
  ```

  Это ссылка на рабочую книгу `Книга1.xls` в коллекции `Workbooks`. Коллекция `Workbooks` находится в объекте `Application`. Переходя на следующий уровень, вы можете сослаться на лист `Лист1` в книге `Книга1.xls`.

  ```
  Application.Workbooks("Книга1.xls").Worksheets("Лист1")
  ```

  Перейдите еще на один уровень ниже, чтобы сослаться на отдельную ячейку.

  ```
  Application.Workbooks("Книга1.xls").Worksheets("Лист1").Range("A1")
  ```

- При опущенной ссылке на объекты Excel по умолчанию используются активные объекты.

  Если книга `Лист1` — активная рабочая книга, то предыдущую ссылку можно упростить.

  ```
  Worksheets("Лист1").Range("A1")
  ```

  Если вы знаете, что лист `Лист1` — активный, то ссылку можно упростить еще больше.

  ```
  Range("A1")
  ```

- Объекты имеют свойства.

  Свойство можно считать *параметром* или *настройкой* объекта. Например, объект диапазона имеет такие свойства, как `Value` (Значение) и `Name` (Имя). Объект диаграммы обладает такими свойствами, как `HasTitle` (Заголовок) и `Type` (Тип). Вы вправе использовать VBA, чтобы задать свойства объектов и изменить их.

  Свойства в программном коде отделяются от названия объекта точкой.

  Например, вы можете сослаться на значение в ячейке `A1` листа `Лист1` следующим образом:

  ```
  Worksheets("Лист1").Range("A1").Value
  ```

Аналогия

Если вы любите аналогии, то это примечание для вас. Возможно, оно поможет понять отношения между объектами, свойствами и методами в VBA. В этой аналогии Excel сравнивается с сетью ресторанов быстрого питания.

Основной элемент Excel — объект `Workbook` (Рабочая книга). В сети ресторанов быстрого питания основным элементом является отдельный ресторан. В Excel вы можете добавлять рабочие книги и закрывать их; все открытые рабочие книги называются `Workbook` (коллекция объектов `Workbooks`). Аналогичным образом руководство сети ресторанов может открывать новые рестораны и закрывать старые — и все они в сети могут рассматриваться как коллекция объектов `Restaurants`.

Рабочая книга Excel является объектом, но она также содержит другие объекты, например, рабочие листы, диаграммы, модули VBA и т.д. Более того, каждый объект в рабочей книге может содержать собственные объекты. Например, объект `Worksheet` (Рабочий лист) включает объекты `Range` (Диапазон), `PivotTable` (Сводная таблица), `Shape` (Форма) и т.д.

Продолжим нашу аналогию: ресторан быстрого питания (как и рабочая книга) содержит свои объекты, например, кухню `Kitchen`, столовое помещение `DiningArea` и `Tables` (Коллекция столов). Кроме того, руководство вправе добавлять или удалять объекты из объекта `Restaurant`. Например, в коллекцию `Tables` можно добавить дополнительные столы. Каждый такой объект иногда содержит другие объекты. Например, объект `Kitchen` включает объект `Stove` (Плита), `VentilationFan` (Вентилятор), `Chef` (Шеф-повар), `Sink` (Раковина) и т.д.

Пока все удачно — аналогия работает. Посмотрим, можно ли продолжить сравнение.

Объекты Excel обладают свойствами. Например, объект `Range` имеет свойства значения `Value` и имени `Name`, а объект `Shape` — свойства ширины `Width`, высоты `Height` и т.д. Объекты в ресторане быстрого питания тоже обладают свойствами. К примеру, объект `Stove` имеет такие свойства, как температуру `Temperature` и количество конфорок `NumberOfBurners`. У объекта `VentilationFan` есть собственный набор свойств (`TurnedOn` (Включен), `RPM` (Частота вращения) и т.д.).

Помимо свойств, объекты Excel также располагают методами, выполняющими операции над объектом. Например, метод `ClearContents` удаляет содержимое объекта `Range`. Объекты ресторана быстрого питания тоже обладают методами. Можно легко представить себе метод `ChangeThermostat` (Изменить температуру) объекта `Stove` или метод `SwitchOn` (Включить) для объекта `VentilationFan`.

В Excel методы иногда используются для изменения свойств объекта. Метод `ClearContents` объекта `Range` изменяет свойство `Value` объекта `Range`. Аналогично, метод `ChangeThermostat` объекта `Stove` изменяет его свойство `Temperature`.

В VBA существует возможность писать процедуры для управления объектами Excel. В ресторане быстрого питания руководство может давать указания по работе с объектами в ресторанах (“Включить плиту и переключить вентилятор на максимальный режим”).

- Вы вправе присваивать значения переменным VBA. Переменную можно считать константой, которая используется для хранения конкретного значения.

  Чтобы присвоить значение ячейки `A1` листа `Лист1` переменной с названием *Interest*, используйте следующий оператор VBA.

  ```
  Interest = Worksheets("Лист1").Range("A1").Value
  ```

- У объектов есть методы.

  Метод — это действие, которое выполняется над объектом. Например, один из методов объекта `Range` — `ClearContents`. Этот метод удаляет содержимое диапазона ячеек.

- Методы вводятся после названия объекта с методом, в роли разделителя выступает точка.

  Например, для удаления содержимого ячейки `A1` активного рабочего листа используется следующая команда.

  ```
  Range("A1").ClearContents
  ```

- VBA также поддерживает конструкции современных языков программирования (в том числе массивы, циклы и т.д.).

Следует отметить, что в этом разделе VBA описан достаточно полно. Теперь вам осталось изучить представленные возможности более подробно, чему и посвящена остальная часть главы.

Знакомство с редактором Visual Basic

Начиная с Excel 97, VBA-модули вводятся с помощью специальной программы, включенной в приложение Excel. Для работы и просмотра модулей VBA используется редактор Visual Basic (VBE — Visual Basic Editor). Он запускается как автоматически — при редактировании макросов, так и отдельно — для создания новых процедур. Редактор VBE обладает всеми необходимыми средствами для управления VBA-кодом в Excel.

НА ЗАМЕТКУ: Модули VBA все еще сохраняются вместе с файлами рабочей книги; просто они не видны до тех пор, пока не запущен редактор VBE.

Запуск VBE

Во время работы в Excel вы можете перейти к окну VBE с помощью одного из следующих способов.

- Нажать <Alt+F11>.
- Выбрать команду Сервис⇨Макрос⇨Редактор Visual Basic.
- Щелкнуть на кнопке Редактор Visual Basic, расположенной на панели инструментов Visual Basic.

НА ЗАМЕТКУ: Не путайте редактор Visual Basic Editor с редактором сценариев Microsoft Script Editor. Это две разные вещи. Редактор сценариев используется для редактирования сценариев HTML, написанных на VBScript или JavaScript. Редактор сценариев в этой книге не рассматривается.

На рис. 7.3 показано окно редактора Visual Basic. Вполне вероятно, что окно VBE в вашем компьютере будет выглядеть не так, как на рисунке. Его настройки легко изменить: существует возможность скрывать панели, изменять их размеры, закреплять, изменять расположение и т.д.

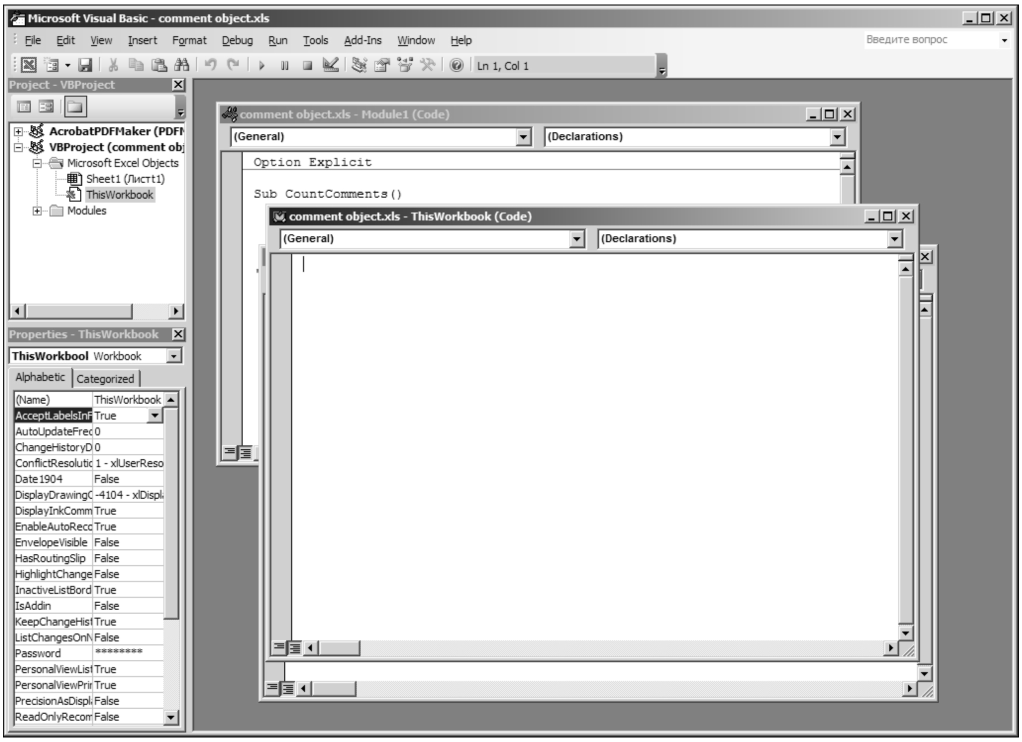


Рис. 7.3. Окно Microsoft Visual Basic

Окна VBE

VBE состоит из ряда элементов. В следующих разделах кратко описаны ключевые компоненты редактора Visual Basic.

СТРОКА МЕНЮ

Строка меню VBE, естественно, работает, как и строка меню любого другого приложения. Она содержит команды, используемые для управления различными компонентами VBE. Кроме того, для выполнения многих команд меню используются комбинации клавиш. Например, для команды View⇨Immediate Window (Вид⇨Окно отладки) применяется комбинация клавиш <Ctrl+G>.

СОВЕТ

В VBE также представлены контекстные меню. Щелкнув правой кнопкой мыши практически на любом элементе окна VBE, вы увидите меню, предлагающее ряд команд.

ПАНЕЛИ ИНСТРУМЕНТОВ

Стандартная панель инструментов Standard, которая по умолчанию находится под строкой меню, — это одна из шести панелей инструментов, используемых в VBE (строка меню тоже считается панелью инструментов). Панели инструментов VBE работают, как и в Excel: вы можете задавать специальные настройки для панелей инструментов, перемещать их, отображать другие панели инструментов и т.д. Для управления панелями инструментов VBE используется команда View⇨Toolbars⇨Customize (Вид⇨Панели инструментов⇨Настройка).

ОКНО PROJECT EXPLORER

В окне Project Explorer отображается древовидная структура всех открытых в данный момент в Excel рабочих книг (включая надстройки и скрытые рабочие книги). Каждая рабочая книга известна как *проект*.

Если в редакторе Visual Basic окно Project Explorer не отображено, нажмите <Ctrl+R>. Чтобы скрыть его, щелкните на кнопке закрытия окна в строке заголовка (или щелкните правой кнопкой мыши в любом месте окна и выберите команду Hide (Скрыть) из контекстного меню).

ОКНО КОДА

Окно кода (которое иногда называют Module) содержит код VBA. Для каждого элемента проекта представлено собственное окно кода. Чтобы просмотреть код объекта, дважды щелкните мышью на этом объекте в окне Project Explorer. Например, чтобы просмотреть код объекта `Лист1`, дважды щелкните на элементе `Лист1` в окне Project Explorer. Если вы не создавали для него VBA-код, открывшееся окно будет пустым.

Существует еще один способ просмотреть код объекта — выделите этот объект в окне Project Explorer, а затем щелкните на кнопке View Code (Просмотр кода) на панели инструментов вверху окна Project Explorer.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Окна кода рассмотрены далее в этой главе в разделе "Работа с окнами кода".

ОКНО ОТЛАДКИ

Окно отладки предназначено для непосредственного выполнения операторов VBA, тестирования операторов и отладки кода. Это окно может отображаться и скрываться. Если окно Immediate в данный момент не отображается на экране, нажмите <Ctrl+G>. Чтобы закрыть окно Immediate, щелкните на кнопке его закрытия в строке заголовка (или щелкните правой кнопкой мыши в любом месте окна и выберите опцию Hide из контекстного меню).

Работа с Project Explorer

При работе в редакторе Visual Basic каждая рабочая книга Excel и открытые в данный момент надстройки рассматриваются как *проекты*. Проект можно считать коллекцией объектов, организованных в виде иерархической структуры. Вы раскроете проект, если щелкнете на знаке "плюс" слева от его названия в окне Project Explorer. Проект сворачивается при щелчке на знаке "минус" слева от его названия. Кроме того, для разворачивания и сворачивания проекта можно использовать кнопку Toggle Folders (Показать папки) на панели инструментов окна Project Explorer. При попытке развернуть проект, защищенный паролем, отображается окно ввода пароля.

На рис. 7.4 показано окно Project Explorer с несколькими проектами (среди них только одна рабочая книга).

ВНИМАНИЕ

При запуске VBE отображаемый программный модуль не всегда соответствует объекту, который выделен в окне Project Explorer. Чтобы убедиться, что вы работаете в нужном программном модуле, дважды щелкните на объекте в окне Project Explorer.

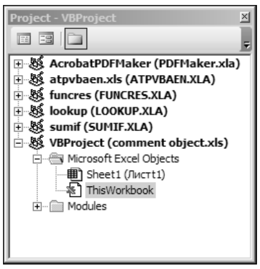


Рис. 7.4. Окно Project Explorer с несколькими проектами

Если в Excel загружено несколько рабочих книг и надстроек, окно Project Explorer будет загромождено. К сожалению, скрыть проекты в окне Project Explorer невозможно. Но если вы детально не рассматриваете отдельные проекты, то последние можно отображать в свернутом виде.

Дерево каждого проекта в развернутом виде имеет как минимум один узел под названием `Microsoft Excel Objects`. В этом узле содержатся элементы каждого рабочего листа и лист диаграмм рабочей книги (рабочий лист считается объектом), а также объект под названием `ЭтаКнига`, представляющий объект `ActiveWorkbook`. Если в проекте используются модули VBA, то в дереве отображается также узел `Modules`, в котором перечислены модули. Проект может также содержать узел `Forms`, содержащий объекты `UserForm` (пользовательские формы, известные как пользовательские диалоговые окна). Если в проекте находятся модули классов, то в дереве отображается узел под названием `Class Modules`.

Добавление нового модуля VBA

Чтобы добавить в проект новый модуль VBA, выделите название проекта в окне Project Explorer и выберите команду Insert⇨Module (Вставка⇨Модуль). Также можно щелкнуть правой кнопкой мыши на названии проекта и выбрать команду Insert⇨Module в контекстном меню.

СОВЕТ При записи макроса Excel автоматически вставляет модуль VBA для хранения записанного кода.

Удаление модуля VBA

Чтобы удалить из проекта модуль VBA или модуль класса, выделите название модуля в окне Project Explorer и используйте команду File⇨Remove *xxx* (где *xxx* — название модуля). Кроме того, вы можете щелкнуть правой кнопкой мыши на названии модуля и выбрать команду Remove *xxx* из контекстного меню. Перед удалением модуля отображается запрос на его экспортирование — детально об этом рассказано в следующем разделе. Вы не сможете удалить программные модули, соответствующие рабочей книге (программный модуль `ЭтаКнига`), а также рабочему листу (например, программный модуль `Лист1`).

Экспорт и импорт объектов

За исключением объектов, перечисленных в узле `References`, каждый объект в проекте можно сохранить в отдельном файле. Сохранение отдельного объекта в проекте называется *экспортом*. Соответственно, вы можете также *импортировать* объекты в проект. Экспорт и импорт объектов востребован, если созданный ранее объект (например, модуль VBA или форму `UserForm`) нужно использовать в другом проекте.

Чтобы экспортировать объект, выберите его в окне Project Explorer и выполните команду File⇨Export File (или нажмите <Ctrl+E>). При этом отображается диалоговое окно, запрашивающее имя файла. Обратите внимание, что сам объект остается в проекте (а экспортируется только его копия). Если вы экспортируете объект `UserForm`, экспортируется также весь код, связанный с формой `UserForm`.

Чтобы импортировать файл в проект, выберите имя проекта в окне Project Explorer и выполните команду File⇨Import File. Появится диалоговое окно, в котором необходимо указать имя файла. Вы можете импортировать только те файлы, которые экспортированы с помощью команды File⇨Export File.

Если вы решили скопировать в другой проект модуль или объект `UserForm`, не обязательно использовать функции экспорта и импорта. Убедитесь, что оба проекта открыты, затем активизируйте окно Project Explorer и перетащите необходимый объект из одного проекта в другой.

Работа с окнами кода

Когда вы в совершенстве овладеете VBA, то будете проводить *много* времени, работая в окнах кода. Каждому объекту в проекте соответствует свое окно кода. Такими объектами могут быть:

- сама рабочая книга (`ЭтаКнига` в окне Project Explorer);
- рабочий лист или лист диаграмм рабочей книги (например, `Лист1` или `Диаграмма1` в окне Project);
- модуль VBA;
- модуль класса (специальный тип модуля, позволяющий создавать новые классы объектов);
- форма `UserForm`.

Сворачивание и восстановление окон

В любой момент в редакторе VBE можно открыть несколько окон кода, и это существенно усложняет работу. Окна кода во многом напоминают окна рабочих листов Excel. Вы можете их свернуть, развернуть, скрыть, изменить порядок отображения на экране и т.д. Многие пользователи предпочитают разворачивать окно кода, над которым они работают в данный момент, что позволяет видеть бо́льшую часть программы, не отвлекаясь на другие модули. Чтобы максимизировать окно кода, щелкните на кнопке Развернуть в строке заголовка или дважды щелкните на самой строке заголовка. Чтобы вернуть окну кода прежний размер, щелкните на кнопке Восстановить в строке заголовка.

Иногда необходимо, чтобы на экране отображалось два или более окон кода. Например, требуется сравнить код в двух модулях или скопировать код из одного модуля в другой.

Сворачивая окно кода, вы скрываете его в окне редактора. Кроме того, вы можете щелкнуть на кнопке Закрыть в строке заголовка окна кода, чтобы полностью его закрыть. Открыть окно кода заново можно, дважды щелкнув на соответствующем объекте в окне Project Explorer.

VBE не позволяет закрывать рабочую книгу. Для этого вы должны вернуться в окно Excel и там закрыть книгу. Однако вы можете использовать окно Immediate, чтобы закрыть рабочую книгу или отключить надстройку. Активизируйте окно Immediate, введите оператор VBA (пример показан ниже) и нажмите <Enter>.

```
Workbooks("myaddin.xla").Close
```

Как видно, этот оператор выполняет метод `Close` объекта `Workbook`, закрывающий рабочую книгу. В данном примере рабочая книга является надстройкой.

Сохранение программы VBA

Как правило, окно кода содержит четыре типа кода.

- *Процедуры.* Это набор инструкций, выполняющих определенное действие.

- *Функции*. Это набор инструкций, возвращающий значение или массив значений (концепция функции VBA подобна такой же функции Excel).
- *Процедуры свойств*. Специальные процедуры, используемые в модулях классов.
- *Объявление* — это информация о переменной, предоставляемая VBA. Например, можно объявить тип данных для переменных, которые вы планируете использовать в коде.

В отдельном модуле VBA может храниться любое количество процедур, функций и объявлений. Способ организации модуля VBA зависит только от вашего желания. Некоторые предпочитают записывать весь код VBA приложения в одном модуле VBA; другие разделяют код на несколько разных модулей.

Терминология

В разных частях книги применяются термины *подпрограмма*, *процедура*, *макрос*. Программисты для описания автоматизированной задачи обычно используют слово *процедура*. В Excel процедуру также называют *макросом*. Технически процедура может быть двух видов: процедура типа `Sub` или процедура функции (или просто функция), оба вида иногда называют *подпрограммами*. В книге эти термины используются как синонимы. Тем не менее, между процедурами типа `Sub` и функциями существует большая разница. О ней речь пойдет в главах 9 и 10.



Несмотря на то, что пользователям предоставляются широкие возможности по определению места хранения кода VBA, существуют некоторые ограничения на его расположение. Процедуры обработки событий должны содержаться в окне кода объекта, которому соответствует это событие. Например, если вы пишете процедуру, которая выполняется при открытии рабочей книги, то эта процедура должна располагаться в окне кода для объекта `ЭтаКнига` и иметь специальное название. Подобный вопрос станет более понятным после того, как вы рассмотрите события (глава 19) и пользовательские формы `UserForm` (часть IV).

Ввод кода VBA

Для того чтобы выполнить одно из действий программным образом, необходимо написать программу VBA в окне кода. Код VBA располагается в процедуре. Процедура состоит из операторов VBA. На данном этапе (для примера) остановимся только на одном типе окна кода — модуле VBA.

Вы можете добавить код в модуль VBA тремя способами.

- Ввести код традиционным способом: с помощью клавиатуры.
- Использовать функцию записи макросов в Excel, чтобы записать действия и преобразовать их в код VBA.
- Скопировать текст программы из другого модуля и вставить его в модуль, над которым работаете.

ВВОД КОДА ВРУЧНУЮ

Иногда самый простой путь является наилучшим. Непосредственное введение кода связано... с использованием клавиатуры, т.е. вы вводите код программы с помощью клавиатуры. Клавиша <Tab> при этом поможет задать отступ в строках, которые логически принадлежат одной группе (например, условные операторы `If` и `End If`).

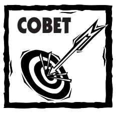
Это совершенно не обязательно, но помогает быстрее освоить программу, анализируя ее блочную структуру. Именно поэтому подобный подход в программировании называется “хорошим стилем”.

Ввод и редактирование кода в модуле VBA выполняется обычным образом. Вы можете выделять текст, копировать, вырезать его, а затем вставлять в другое место программы.

Отдельная инструкция в VBA может иметь произвольную длину. Однако для обеспечения удобочитаемости кода длинные инструкции лучше разбить на две или более строк. Для этого следует в конце строки ввести пробел и символ подчеркивания, а затем нажать <Enter> и продолжить инструкцию в следующей строке. Например, ниже приведен один оператор VBA, разбитый на четыре строки.

```
MsgBox "Невозможно найти" & UCase(SHORTCUTMENUFILE) _
     & vbCrLf & vbCrLf & "Файл должен находиться в _
     " & ThisWorkbook.Path & vbCrLf & vbCrLf & _
     "Возможно, требуется переустановить BudgetMan", vbCritical, APPNAME
```

Обратите внимание, что три последние строки этого оператора введены с отступом. Это необязательное условие, однако таким образом вы указываете, что на самом деле эти четыре строки являются одним оператором.



Как и в Excel, в VBE существует несколько уровней отмены выполненных операций. Поэтому, если вы по ошибке удалили инструкцию, можете несколько раз щелкнуть на кнопке Undo (Отменить) или нажать <Ctrl+Z>, и инструкция вновь появится в коде. После отмены операции можно щелкнуть на кнопке Redo (Вернуть), чтобы вернуть изменения, которые ранее были отменены. Эта функция поможет исправить критически важные ошибки, поэтому не пренебрегайте ее использованием.

Выполните такие действия: добавьте в проект модуль VBA, затем введите следующую процедуру в окне кода данного модуля.

```
Sub SayHello()
     Msg = "Ваше имя " & Application.UserName & "?"
     Ans = MsgBox(Msg, vbYesNo)
     If Ans = vbNo Then
          MsgBox "Ничего страшного."
     Else
          MsgBox "Наверное, я ясновидящий!"
     End If
End Sub
```

На рис. 7.5 показано, как это выглядит в модуле VBA.

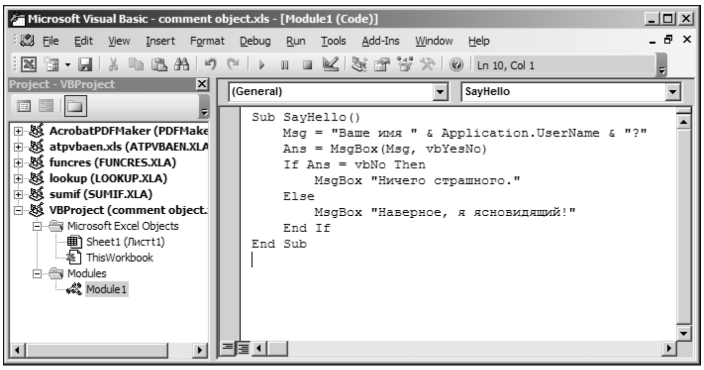


Рис. 7.5. Ваша первая процедура VBA

При вводе кода вы могли заметить, что VBE вносит некоторые изменения во введенный текст. Например, если вы пропустите пробел перед или после знака равенства (=), VBE вставит его автоматически. Кроме того, изменяется цвет некоторых слов кода. Это нормально, и позже вы данный факт оцените.

При выполнении процедуры `SayHello` убедитесь, что курсор находится во введенном вами тексте. Затем выполните одно из следующих действий.

1. Нажмите <F5>.
2. Выберите команду Run⇨Run Sub/UserForm.
3. Щелкните на кнопке Run Sub/UserForm на стандартной панели инструментов.

Если вы ввели код правильно, процедура будет выполнена. Вы сможете выбрать ответ в простом диалоговом окне (рис. 7.6), в котором отображается имя пользователя, заданное в диалоговом окне Параметры. Это окно вызывается из меню Сервис программы Excel. Обратите внимание, что при выполнении макроса активизируется программа Excel. На данном этапе вам не обязательно понимать принципы работы программы; в этом вы разберетесь, прочтя следующие разделы и главы настоящей книги.

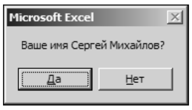


Рис. 7.6. Результат выполнения процедуры, показанной на рис. 7.6

В большинстве случаев вы будете запускать макросы в Excel. Однако тестировать макрос удобнее, выполняя его прямо в VBE.

Написанная только что программа представляет собой процедуру VBA (называемую также *макросом*). Когда вы даете команду выполнить макрос, VBE быстро компилирует код и запускает его. Другими словами, каждая инструкция анализируется в VBE, а Excel всего лишь выполняет то, что указано в инструкциях. Вы можете выполнить макрос сколько угодно раз, хотя через некоторое время он теряет свою привлекательность.

При построении в этой простой процедуре использовались следующие принципы (все они рассмотрены далее в главе).

- Объявление процедуры (первая строка).
- Присвоение значения переменным (`Msg` и `Ans`).
- Конкатенация строк (с помощью оператора `&`).
- Использование встроенной функции VBA (`MsgBox`).
- Применение встроенных констант VBA (`vbYesNo` и `vbNo`).
- Использование конструкции `If-Then-Else`.
- Окончание процедуры (последняя строка).

Неплохо для первого раза, не правда ли?

ИСПОЛЬЗОВАНИЕ ФУНКЦИИ ЗАПИСИ МАКРОСОВ

Одним из способов создания кода модуля VBA является запись последовательности действий с помощью специальной функции записи макросов Excel.

Как бы вы ни старались, но записать показанную в предыдущем примере процедуру `SayHello` вы не сможете. Запись макросов — полезное средство, но оно имеет свои ограничения. После записи макросов вы неоднократно будете вносить изменения или вводить дополнительный код вручную.

В следующем примере показано, как записать макрос, изменяющий ориентацию страницы на альбомную. Если вы хотите получить его самостоятельно, то начните работу с пустой рабочей книги и выполните следующие действия.

1. Активизируйте рабочий лист в книге (подойдет любой лист).
2. Выберите команду Сервис⇨Макрос⇨Начать запись.
3. При этом Excel отображает диалоговое окно Запись макроса.
4. Щелкните на кнопке ОК, чтобы принять параметры по умолчанию.
5. Excel автоматически вставит новый модуль VBA в проект. Начиная с этого момента, Excel будет преобразовывать ваши действия в код VBA. При записи в строке состояния отображается слово Запись, кроме того, в окно добавляется небольшая плавающая панель инструментов, содержащая две кнопки (Остановить запись и Относительная ссылка).
6. Выполните команду Файл⇨Параметры страницы.
7. Excel отобразит диалоговое окно Параметры страницы.
8. Выберите переключатель Альбомная и щелкните на кнопке ОК, чтобы закрыть диалоговое окно.
9. Щелкните на кнопке Остановить запись на панели инструментов (или выберите Сервис⇨Макрос⇨Остановить запись).
10. Excel прекратит записывать ваши действия.

Чтобы просмотреть макрос, запустите VBE (проще всего нажать <Alt+F11>) и найдите проект в окне Project Explorer. Щелкните на узле Modules, чтобы развернуть его. Затем щелкните на элементе Module1, чтобы отобразить окно кода (если в проекте уже присутствовал модуль Module1, новый макрос будет находиться в модуль Module2). Код, созданный одной командой, представлен в листинге 7.1. Если вы используете не Excel 2003, а иную версию, текст программы может немного отличаться.

Листинг 7.1. Макрос изменения ориентации страницы на альбомную

```
Sub Макрос1()
'
' Макрос1 Макрос
' Макрос записан 19.08.2003
'

'
    With ActiveSheet.PageSetup
        .PrintTitleRows = ""
        .PrintTitleColumns = ""
    End With
    ActiveSheet.PageSetup.PrintArea = ""
    With ActiveSheet.PageSetup
        .LeftHeader = ""
        .CenterHeader = ""
        .RightHeader = ""
        .LeftFooter = ""
```

```
        .CenterFooter = ""
        .RightFooter = ""
        .LeftMargin = Application.InchesToPoints(0.787401575)
        .RightMargin = Application.InchesToPoints(0.787401575)
        .TopMargin = Application.InchesToPoints(0.984251969)
        .BottomMargin = Application.InchesToPoints(0.984251969)
        .HeaderMargin = Application.InchesToPoints(0.5)
        .FooterMargin = Application.InchesToPoints(0.5)
        .PrintHeadings = False
        .PrintGridlines = False
        .PrintComments = xlPrintNoComments
        .PrintQuality = 1200
        .CenterHorizontally = False
        .CenterVertically = False
        .Orientation = xlLandscape
        .Draft = False
        .PaperSize = xlPaperA4
        .FirstPageNumber = xlAutomatic
        .Order = xlDownThenOver
        .BlackAndWhite = False
        .Zoom = 100
        .PrintErrors = xlPrintErrorsDisplayed
    End With
End Sub
```

Возможно, вас удивит количество кода, генерированного всего лишь одной командой (особенно если вы записываете макрос впервые). Несмотря на то, что вы изменили только одну простую настройку в диалоговом окне Параметры страницы, Excel генерирует код, задающий *все* параметры в этом диалоговом окне.

Таким образом, зачастую программа, полученная при записи макроса, избыточна. Если вы хотите, чтобы макрос всего лишь изменял ориентацию страницы на альбомную, то можно значительно упростить его, удалив ненужный код. Это облегчит восприятие макроса и ускорит его выполнение, поскольку избавит его от лишних операций. Упростить макрос вы вправе до следующего вида.

```
Sub Макрос1
     With ActiveSheet.PageSetup
          .Orientation = xlLandscape
     End With
End Sub
```

Мы удалили весь код, кроме строки, изменяющей свойство `Orientation`. На самом деле данный макрос можно упростить еще больше, так как конструкция `With-End With` не обязательна при изменении только одного свойства.

```
Sub Макрос1
  ActiveSheet.PageSetup.Orientation = xlLandscape
End Sub
```

В данном примере макрос изменяет свойство `Orientation` объекта `PageSetup` активного листа. Отметим, что `xlLandscape` — это заранее заданная константа, которая предназначена для изменения ориентации страницы. Переменная `xlLandscape` имеет значение 2, а `xlPortrait` — значение 1. Следующий макрос работает, как и предыдущий `Макрос1`.

```
Sub Макрос1
  ActiveSheet.PageSetup.Orientation = 2
End Sub
```

Многим пользователям легче запомнить название константы, чем произвольные числа. Вы можете воспользоваться справочной системой, чтобы выучить соответствующие каждой настройке константы.

Зачастую данная процедура вводится непосредственно в модуль VBA, но для этого необходимо знать, какие объекты, свойства и методы требуется использовать. Но ведь записать макрос быстрее. Кроме того, данный пример продемонстрировал наличие у объекта `PageSetup` свойства `Orientation`.

Я считаю, что запись действий — это *лучший* способ изучить VBA. Если у вас возникают проблемы с введением кода, то воспользуйтесь функцией записи действий. Даже если вы можете получить не совсем то, что ожидалось, результирующий код укажет правильное направление. Для получения информации об объектах, свойствах и методах, которые присутствуют в записанном коде, используйте электронную справочную систему.

Механизм записи макросов подробнее рассмотрен далее в этой главе. (См. раздел “Функция записи макросов”.)

КОПИРОВАНИЕ КОДА VBA

Выше были рассмотрены способы непосредственного ввода кода и записи действий для создания программы VBA. Последний метод добавления кода в модуль VBA — копирование текста программы из другого модуля. Например, вы могли написать процедуру в одном из более ранних проектов, которая также используется в текущем проекте. Вместо того, чтобы заново вводить код, достаточно открыть рабочую книгу, активизировать модуль и использовать стандартные способы копирования и вставки, чтобы скопировать его в текущий модуль VBA. Если после вставки возникает необходимость, подкорректируйте код модуля.

Как уже отмечалось в этой главе, вы также можете импортировать в файл модуль, который был ранее экспортирован.

Настройка среды VBE

В процессе программирования в Excel вы будете проводить много времени, работая в окнах VBE. Чтобы сделать редактор более удобным, вы можете настроить некоторые его параметры.

В строке меню окна VBE выберите команду Tools⇨Options (Сервис⇨Параметры). Появится диалоговое окно с четырьмя вкладками: Editor (Редактор), Editor Format (Формат редактора), General (Общие) и Docking (Присоединение). Некоторые наиболее часто используемые параметры этих вкладок будут рассмотрены в следующих разделах. Кстати, не путайте это окно с диалоговым окном Параметры программы Excel, которое можно открыть в Excel с помощью подобной команды — Сервис⇨Параметры. В этих диалоговых окнах, хотя они и называются одинаково, задаются разные параметры.

Вкладка Editor

На рис. 7.7 показаны параметры, доступные на вкладке Editor диалогового окна Options.

ПАРАМЕТР AUTO SYNTAX CHECK

Настройка Auto Syntax Check (Автоматическая проверка синтаксиса) определяет, будет ли появляться диалоговое окно, когда VBE обнаруживает синтаксическую

ошибку в коде VBA. В этом диалоговом окне указывается тип допущенной ошибки. Если снять этот флажок, то VBE выделит синтаксические ошибки, отобразив соответствующие фрагменты кода другим цветом, и вам не придется работать в диалоговых окнах, которые появляются на экране.

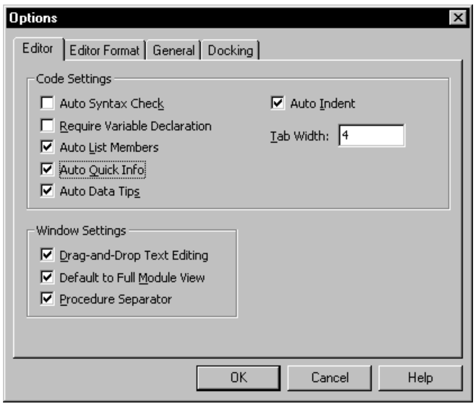


Рис. 7.7. Вкладка Editor *диалогового окна* Options

Некоторые пользователи отключают параметр Auto Syntax Check, если они без труда могут определить, в чем заключается ошибка. Но если вы только начинаете работать с VBA, то дополнительная помощь может оказаться весьма полезной.

ПАРАМЕТР REQUIRE VARIABLE DECLARATION

При установленном параметре Require Variable Declaration (Обязательное декларирование переменных) VBE вставляет в начале каждого нового модуля следующий оператор.

```
Option Explicit
```

Если в модуле задан этот оператор, то вы должны явно определить каждую используемую в нем переменную. Таким образом, у вас вырабатывается хорошая привычка, которая, правда, требует от вас дополнительных усилий. Если вы не объявляете переменные, все они имеют тип данных `Variant`; это достаточно гибко, но неэффективно с точки зрения использования аппаратных ресурсов и скорости выполнения кода (см. далее в этой главе).

Изменение параметра Require Variable Declaration затрагивает только новые модули, а не уже существующие.

ПАРАМЕТР AUTO LIST MEMBERS

Если выставлена опция Auto List Members (Автоматическая вставка объектов), VBE предоставляет помощь при вводе кода VBA, отображая список элементов текущего объекта. К этим элементам относятся методы и свойства объекта, название которого вводится вручную.

Данный параметр весьма полезен, поэтому его рекомендуется всегда активизировать. На рис. 7.8 показан пример использования опции Auto List Members (предназначение которой станет понятнее, когда вы начнете вводить код VBA самостоятельно). В данном примере VBE отображает список элементов объекта `Application`. Вы можете выбрать элемент из списка, чтобы не вводить его с помощью клавиатуры (в результате название элемента точно будет введено без ошибок).

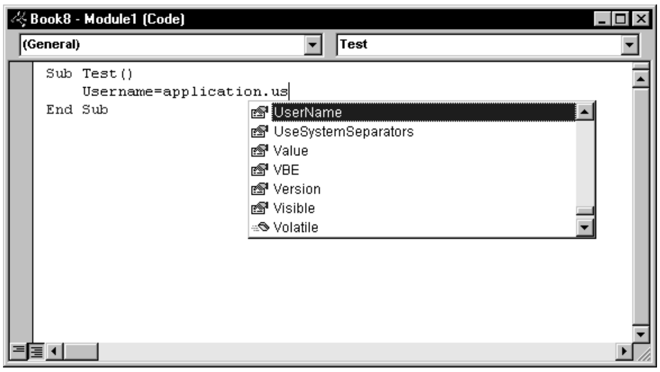


Рис. 7.8. Пример использования опции Auto List Members

ПАРАМЕТР AUTO QUICK INFO

Если включен параметр Auto Quick Info (Отображать краткие сведения), VBE будет отображать информацию об аргументах функций, свойств и методов, названия которых вы вводите с клавиатуры. Это очень полезно, поэтому рекомендуется всегда оставлять эту настройку включенной. На рис. 7.9 данная функция показана в действии: отображается синтаксис свойства `Range`.

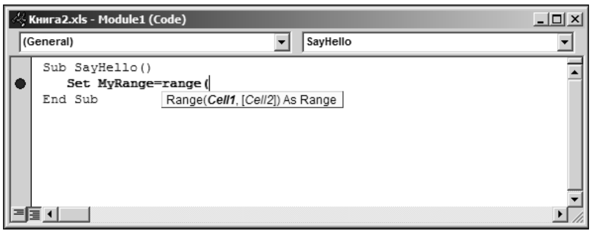


Рис. 7.9. Пример функции Auto Quick Info, предоставляющей сведения о свойстве `Range`

ПАРАМЕТР AUTO DATA TIPS

Если включен параметр Auto Data Tips, VBE отображает при отладке кода значение переменной, над которой находится указатель мыши. Когда вы ознакомитесь с инструментами отладки в VBE, то по достоинству сможете оценить этот параметр. Рекомендуем всегда держать его включенным.

ПАРАМЕТР AUTO INDENT

Настройка Auto Indent (Автоматический отступ) определяет, располагает ли автоматически VBE каждую новую строку программы с тем же отступом, который задан для предыдущей строки. Тем, кто использует отступы в программных кодах, советуем всегда обращаться к этому параметру. Вы можете также задать количество символов в отступе (по умолчанию указано значение 4).

СОВЕТ

Используйте клавишу <Tab>, а не пробел, чтобы задать отступ в коде. Кроме того, для отмены отступа в конкретной строке можно применить комбинацию клавиш <Shift+Tab>, которая работает также и при выделении более чем одного оператора.

ПАРАМЕТР DRAG-AND-DROP TEXT EDITING

При выборе параметра Drag-and-Drop Text Editing (Включить редактирование перетаскиванием) вы можете копировать и перемещать текст, перетаскивая его с помощью мыши. Например, я оставляю этот параметр включенным, но никогда не пользуюсь функцией перетаскивания, так как предпочитаю для копирования и вставки обращаться к комбинациям клавиш.

ПАРАМЕТР DEFAULT TO FULL MODULE VIEW

Параметр Default to Full Module View (По умолчанию использовать полный режим просмотра) определяет принцип просмотра процедуры. Если он включен, процедуры в окне кода помещаются в одно окно с полосой прокрутки. Если же он отключен, то вы можете просмотреть в определенный момент только одну процедуру. Рекомендуем активизировать этот параметр.

ПАРАМЕТР PROCEDURE SEPARATOR

Когда параметр Procedure Separator (Разделение процедур) включен, в конце каждой процедуры в окне кода отображаются специальные разделители. Если вам нравятся эти визуальные подсказки окончания процедуры, выставляйте данный флажок.

Вкладка Editor Format

На рис. 7.10 показана вкладка Editor Format диалогового окна Options.

ПАРАМЕТР CODE COLORS

Параметр Code Colors (Цвета кода) предоставляет возможность выбрать цвета кода (текста и фона) и индикатора, который используется для выделения разных элементов программы VBA. Цвета, конечно, выбираются в зависимости от личных предпочтений. Вы можете согласиться с цветами, принятыми по умолчанию. Однако для разнообразия можете изменить эти настройки.

ПАРАМЕТР FONT

Параметр Font (Шрифт) предоставляет возможность указать шрифт, используемый в модулях VBA. Наибольшая эффективность достигается при работе с моноширинным шрифтом (например, Courier New). В таком шрифте все символы имеют одинаковую ширину, что делает программу более удобной для восприятия и анализа, так как все символы одинаково выровнены, кроме того, хорошо видны пробелы между словами.

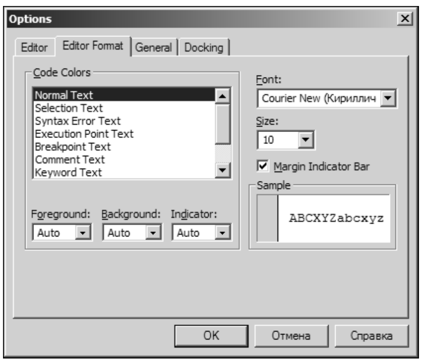


Рис. 7.10. Вкладка Editor Format диалогового окна Options

СПИСОК SIZE

Список Size (Размер) определяет размер шрифта кода модулей VBA. Эта настройка зависит от личных предпочтений, которые, в свою очередь, определяются разрешением монитора и вашим зрением. По умолчанию размер задан равным 10.

ПАРАМЕТР MARGIN INDICATOR BAR

Этот параметр отображает вертикальную полосу вдоль левой границы окна кода, на которой высвечиваются всевозможные индикаторы. Его необходимо выставить; в противном случае вы не увидите полезные графические извещения при отладке кода.

Вкладка General

На рис. 7.11 показаны параметры, доступные на вкладке General (Общие) диалогового окна Options. Практически во всех случаях идеально использовать настройки по умолчанию.



Раздел Error Trapping (Захват ошибок) определяет, что происходит при возникновении ошибки. Если вы создаете процедуры обработки ошибок, то убедитесь, что включен переключатель Break on Unhandled Errors (Остановка при возникновении неисправимой ошибки). При заданном параметре Break on All Errors (Остановка при возникновении любой ошибки) процедуры обработки ошибок игнорируются (а вы вряд ли этого хотите). Методы обработки ошибок детально описываются в главе 9.

Вкладка Docking

На рис. 7.12 показана вкладка Docking диалогового окна Options. Ее параметры определяют поведение нескольких окон редактора VBE — отображаются окна, которые могут быть прикреплены. Когда окно прикреплено, оно фиксируется по отношению к одной из границ окна VBE. В результате намного легче найти вспомогательное окно, так как оно отображается в строго определенной области. Если вы отключите все параметры прикрепления, то окна перемешаются между собой, а это усложнит работу. Как правило, идеальным выбором будут настройки по умолчанию.

Для прикрепления окна просто перетащите его в новое место. Например, вам может понадобиться присоединить окно Project Explorer к левой границе окна. Захватите его за заголовок и переместите влево. Отпустите кнопку мыши в момент, когда окно "прилипнет" к левому краю экрана.

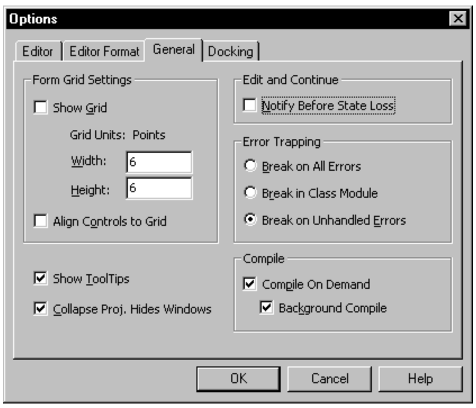


Рис. 7.11. Вкладка General диалогового окна Options

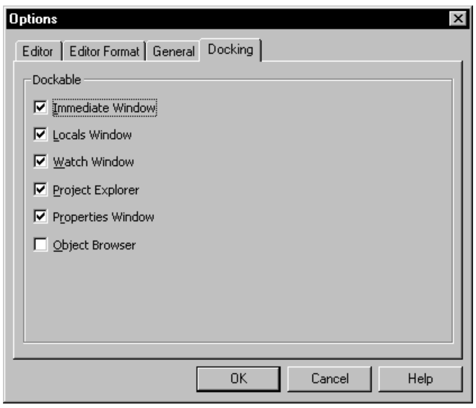


Рис. 7.12. Вкладка Docking диалогового окна Options

Присоединение окон в VBE всегда было основной проблемой при настройке рабочей среды. Очень часто окна отказываются присоединяться. К сожалению, я не знаю, как заставить их работать корректно: после многочисленных попыток вам все же удастся выполнить все правильно, но как быстро это произойдет — не знает никто.

Функция записи макросов

Операция записи макросов уже рассматривалась в одном из предыдущих разделов — это механизм, преобразующий действия в Excel в программу VBA. В этом разделе запись макросов будет описана подробнее.

На панели инструментов Visual Basic в Excel находится несколько полезных для вас кнопок. Среди них — Выполнить макрос, Записать макрос, Остановить запись и Редактор Visual Basic.

Запись макросов — это *чрезвычайно* полезный инструмент, но не забывайте о следующих моментах.

- Запись лучше всего выполнять для простых макросов или небольшого фрагмента более сложного макроса.
- Команда записи макросов не может генерировать программы, которые включают циклические структуры (т.е. повторяющиеся операторы), а также присваивают переменные, выполняют условные операторы, отображают диалоговые окна и т.д.
- Созданная в результате записи программа зависит от определенных вами настроек.
- Вам придется часто дорабатывать записанный код, чтобы удалить лишние команды.

Что записывается

Как вы знаете, функция записи макросов Excel преобразует действия, выполненные с помощью мыши и клавиатуры, в код VBA. Принцип выполнения можно описать на нескольких страницах, но целесообразнее рассмотреть пример. Выполните следующие действия.

1. Начните с пустой рабочей книги.
2. Убедитесь, что окно Excel полностью не развернуто. Добейтесь, чтобы на экране оставалось свободное место.
3. Нажмите <Alt+F11>, чтобы запустить окно VBE, и убедитесь, что *его* окно также не максимизировано.
4. Измените размер и разместите окна Excel и VBE так, чтобы они были видны. (Лучше всего при этом свернуть окна других неиспользуемых приложений.)
5. Активизируйте Excel, выберите команду Сервис⇨Макрос⇨Начать запись и щелкните на кнопке OK, чтобы запустить функцию записи макросов.
6. Excel добавляет новый модуль (под названием `Module1`) и сохраняет его в текущем листе.
7. Перейдите к окну VBE.
8. В окне Project Explorer дважды щелкните на `Module1`, чтобы отобразить содержимое модуля в окне кода.
9. Закройте окно Project Explorer, чтобы освободить место под окно кода.

Окно редактора Visual Basic будет выглядеть, как в примере на рис. 7.13. Размер окон зависит от разрешения экрана.

Теперь поработайте на рабочем листе, выбирая разные команды Excel. Посмотрите, как генерируется код в окне, представляющем модуль VBA. Вам следует выполнить несколько действий: выделить ячейки, ввести данные, изменить формат ячеек. Далее используйте меню и панели инструментов, создайте диаграмму, поработайте с графическими объектами и т.д. Все прояснится, когда на ваших глазах появится код программы.

Относительный или абсолютный?

При записи последовательности действий Excel обычно использует абсолютные ссылки на ячейки. Другими словами, при выделении ячейки Excel делает ее активной (а не исходную ячейку, активную при запуске программы). Проверим на примере, как это работает. Выполните следующие действия и проанализируйте полученный результат.

1. Активизируйте рабочий лист и запустите функцию записи макросов.
2. Активизируйте ячейку `B1`.
3. Введите в ячейку `B1` **Янв**.
4. Перейдите в ячейку `C1` и введите **Фев**.
5. Продолжайте этот процесс, пока в ячейках `B1:G1` не будут введены аббревиатуры первых шести месяцев года.

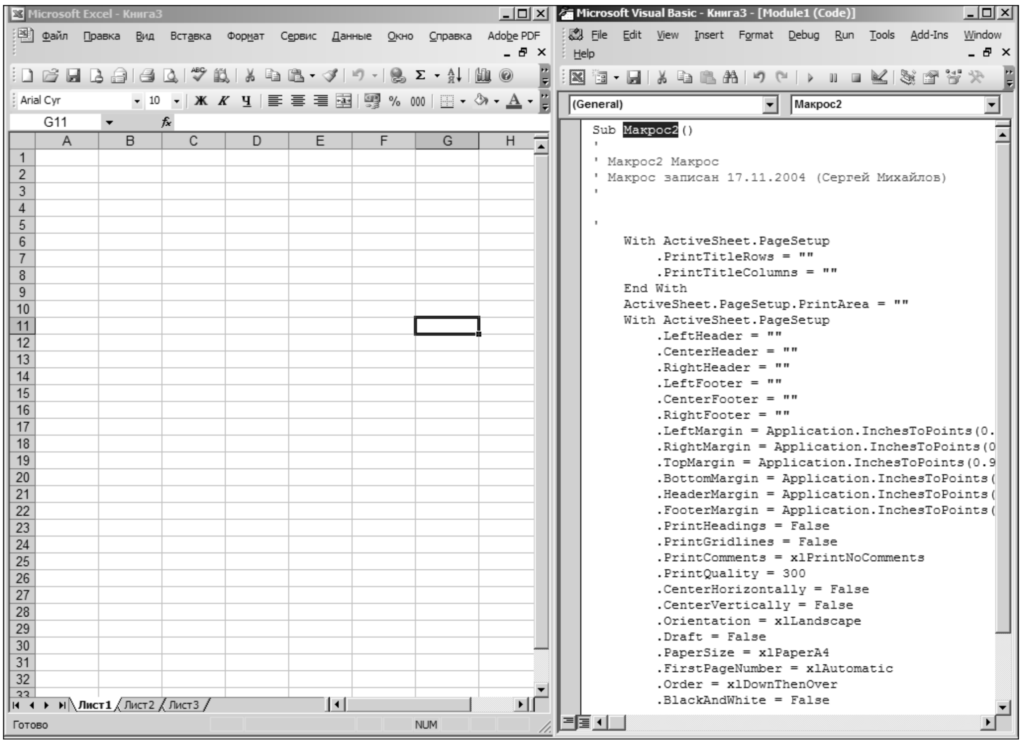


Рис. 7.13. Удобное расположение окон для наблюдения за записью макросов

6. Щелкните на ячейке B1, чтобы снова сделать ее активной.
7. Остановите запись макроса.

Excel генерирует следующий код.

```
Sub Macro()
      Range("B1").Select
      ActiveCell.FormulaR1C1 = "Янв"
      Range("C1").Select
      ActiveCell.FormulaR1C1 = "Фев"
      Range("D1").Select
      ActiveCell.FormulaR1C1 = "Мар"
      Range("E1").Select
      ActiveCell.FormulaR1C1 = "Апр"
      Range("F1").Select
      ActiveCell.FormulaR1C1 = "Май"
      Range("G1").Select
      ActiveCell.FormulaR1C1 = "Июн"
      Range("B1").Select
End Sub
```

Чтобы выполнить этот макрос, обратитесь к команде Сервис⇨Макрос⇨Макросы (или нажмите <Alt+F8>), выберите Макрос1 (или название записанного макроса) и щелкните на кнопке Выполнить.

Макрос при выполнении вновь выполнит действия, записанные ранее. Действия выполняются независимо от того, активны соответствующие ячейки на рабочем листе или нет. При записи макроса с использованием абсолютных ссылок вы всегда будете получать одни и те же результаты.

Однако в некоторых случаях требуется, чтобы записанный макрос работал с *относительными* адресами ячеек. Например, такой макрос обычно вводит названия месяцев в активной ячейке. В таком случае для записи макроса используется относительная форма записи.

При записи макроса отображается панель инструментов Остановить запись, состоящая только из двух кнопок. Вы можете изменить способ записи действий Excel, щелкнув на кнопке Относительная ссылка на панели инструментов Остановить запись. Это кнопка-переключатель. Когда она находится в нажатом состоянии, используется относительный режим записи. Если кнопка — в нормальном состоянии, то выполняется запись в абсолютном режиме. Вы можете изменить режим записи в любое необходимое вам время, даже в процессе выполнения записываемых операций.

Чтобы увидеть этот процесс, вначале следует очистить ячейки в диапазоне B1:D1, а затем выполнить следующие действия.

1. Перейдите к ячейке B1.
2. Выберите Сервис⇨Макрос⇨Начать запись.
3. Щелкните на кнопке ОК, чтобы начать запись.
4. Щелкните на кнопке Относительная ссылка (на панели инструментов Остановить запись), чтобы изменить режим записи на относительный.
5. При щелчке кнопка переходит в нажатое состояние.
6. Введите названия первых шести месяцев года в ячейки B1:G1 (как в предыдущем примере).
7. Выберите ячейку B1.
8. Остановите запись макроса.

Если установить относительный режим записи, созданный Excel код приобретет иной вид.

```
Sub Macro2()
     ActiveCell.FormulaR1C1 = "Янв"
     ActiveCell.OffSet(0, 1).Range("A1").Select
     ActiveCell.FormulaR1C1 = "Фев"
     ActiveCell.OffSet(0, 1).Range("A1").Select
     ActiveCell.FormulaR1C1 = "Мар"
     ActiveCell.OffSet(0, 1).Range("A1").Select
     ActiveCell.FormulaR1C1 = "Апр"
     ActiveCell.OffSet(0, 1).Range("A1").Select
     ActiveCell.FormulaR1C1 = "Май"
     ActiveCell.OffSet(0, 1).Range("A1").Select
     ActiveCell.FormulaR1C1 = "Июн"
     ActiveCell.OffSet(0, 1).Range("A1").Select
End Sub
```

Вы можете выполнить этот макрос, активизировав рабочий лист и выбрав команду Сервис⇨Макрос. Укажите название макроса и щелкните на кнопке Выполнить.

Как вы заметили, в рассматриваемом примере процедура была незначительно изменена (мы активизировали начальную ячейку *перед* началом записи). Это важная операция при записи макроса, использующего в качестве основы активную ячейку.

На первый взгляд, макрос выглядит несколько сложно, хотя на самом деле он довольно прост. Первый оператор вводит **Янв** в активную ячейку. (Используется активная ячейка, так как перед оператором не указан оператор, активизации ячейки.) Следующий оператор использует свойство `Offset` для перемещения курсора на одну ячейку вправо. Следующий оператор вставляет в нее текст и т.д. В отличие от предыдущего, данный макрос всегда начинает ввод текста в активной ячейке.

Вы наверняка заметили, что в этом макросе сгенерирован код, который будто бы ссылается на ячейку `A1` — это может показаться странным, так как ячейка `A1` в данном макросе не задействована. Это побочный эффект функции записи макросов. (Свойство `Offset` рассмотрено далее в настоящей главе.) На данном этапе вам необходимо знать, что макрос работает, как требуется.

Кстати, код, сгенерированный Excel, намного сложнее, чем необходимо, и представляет не самый эффективный способ программирования описанной операции. Следующий макрос, который был введен вручную, представляет собой более простой и быстрый способ выполнить те же действия. В примере показано, что не обязательно выделять ячейку перед помещением в нее информации — это важный момент, который существенно ускоряет работу макроса.

```
Sub Macro3()
     ActiveCell.Offset(0, 0) = "Янв"
     ActiveCell.Offset(0, 1) = "Фев"
     ActiveCell.Offset(0, 2) = "Мар"
     ActiveCell.Offset(0, 3) = "Апр"
     ActiveCell.Offset(0, 4) = "Май"
     ActiveCell.Offset(0, 5) = "Июн"
End Sub
```

Данный макрос можно еще более упростить, используя конструкцию `With-End With`.

```
Sub Macro4()
     With ActiveCell
     .Offset(0, 0) = "Янв"
     .Offset(0, 1) = "Фев"
     .Offset(0, 2) = "Мар"
     .Offset(0, 3) = "Апр"
```

```
    .Offset(0, 4) = "Май"
    .Offset(0, 5) = "Июн"
End Sub
```

Если же вы гениальный программист на VBA (как и я), то можете поразить своих коллег, выполнив все описанное выше в одном операторе.

```
Sub Macro54()
    ActiveCell.Resize( ,6)=Array("Янв", "Фев", "Мар", "Апр", "Май", "Июн")
End Sub
```

Помните, что функция записи макросов имеет два разных режима, поэтому всегда отдавайте отчет, в каком режиме вы работаете. В противном случае результат будет далеко не таким, как вы хотите.

Параметры записи

При записи действий с целью создания кода VBA вы можете настроить несколько параметров. Как вы помните, команда Сервис⇨Макрос⇨Начать запись отображает перед началом записи диалоговое окно Запись макроса. В этом диалоговом окне предоставлено достаточно много данных о макросе. В следующих разделах описаны все параметры указанного окна.

ИМЯ МАКРОСА

Вам предоставляется возможность ввести название записываемой процедуры. По умолчанию Excel использует названия `Макрос1`, `Макрос2` и т.д. для каждого записываемого макроса. Вы можете использовать имя по умолчанию и изменять это название позже. Однако лучше сразу назвать макрос правильным именем.

СОЧЕТАНИЕ КЛАВИШ

Параметр Сочетание клавиш позволяет выполнить макрос с помощью комбинации клавиш. Например, вводя в данном поле **w** (в нижнем регистре), вы можете выполнить макрос — для этого нажмите комбинацию клавиш <Ctrl+W>. При вводе символа **W** (в верхнем регистре) макрос запускается по нажатию комбинации <Ctrl+Shift+W>.

Вы вправе в любой момент добавить или изменить комбинацию клавиш, поэтому необязательно задавать параметр при записи макроса.

СОХРАНИТЬ В

Параметр Сохранить в указывает Excel, где должен храниться макрос, который записывается. По умолчанию Excel помещает записанный макрос в модуле активной рабочей книги. По желанию вы можете записать его либо в новой рабочей книге (Excel открывает пустую рабочую книгу), либо в личной книге макросов.

Личная книга макросов

Если вы создаете макросы VBA, которые считаете особенно полезными, то сохраните их в личной книге макросов для дальнейшего использования. Это рабочая книга под названием `Personal.xls`, которая хранится в папке `Xlstart`. При записи макроса одним из вариантов является запись в личной книге макросов. Файл `Personal.xls` появится лишь тогда, когда вы запишете в него хотя бы один макрос.

ОПИСАНИЕ

По умолчанию Excel вставляет пять строк комментария (три из которых пустые). В них указывается имя макроса, имя пользователя и дата создания. Вы можете добавить в это поле любую информацию, а можете не вводить ни единого слова. Рекомен-

дуем вообще не вводить никаких данных, поскольку впоследствии их потребуется удалять в модуле.

В версиях Excel, более ранних, чем Excel 97, в диалоговом окне Запись макроса присутствовал параметр, позволяющий присваивать макросу команду в меню Сервис. В новых версиях Excel если требуется, чтобы макрос выполнялся из меню, то данная настройка устанавливается дополнительно (см. главу 23).

Улучшение записанных макросов

Итак, вы уже знаете о том, что запись действий при выполнении всего лишь одной команды (Файл⇨Параметры страницы) приводит к генерированию большого объема кода VBA. Во многих случаях записанный код включает ненужные команды, которые следует удалять вручную.

Кроме того, функция записи макросов не всегда гарантирует получение эффективного кода. Проанализировав созданную программу, вы увидите следующее: как правило, Excel анализирует, *что* выделено (т.е. определяет активный объект), и затем использует в генерируемых операторах объект `Selection`. Далее представлен пример записи при выделении диапазона ячеек, использовании кнопок на панели инструментов Форматирование (с целью изменить числовой формат) и применении полужирного и курсивного начертания.

```
Range("A1:C5").Select
Selection.NumberFormat = "#,##0.00"
Selection.Font.Bold = True
Selection.Font.Italic = True
```

СОВЕТ

Если вы используете для записи этого макроса диалоговое окно Формат, то обнаружите, что Excel записывает очень много лишнего. Запись щелчков на кнопках панели инструментов часто дает более эффективный результат.

Мы описали лишь один способ выполнения указанных действий. Вы можете также использовать более эффективную конструкцию `With-End With`.

```
Range("A1:C5").Select
With Selection
     .NumberFormat = "#,##0.00"
     .Font.Bold = True
     .Font.Italic = True
End With
```

Можно избежать применения метода `Select` и сделать код еще лучше.

```
With Range("A1:C5")
     .NumberFormat = "#,##0.00"
     .Font.Bold = True
     .Font.Italic = True
End With
```

Если в вашем приложении важна скорость выполнения операций, то необходимо тщательно анализировать любой записанный код VBA, чтобы сделать его как можно более эффективным.

Конечно, вам придется хорошо разобраться в VBA, прежде чем приступить к улучшению кода записанных макросов. А пока примите к сведению, что записанный код VBA не всегда является наилучшим и наиболее эффективным решением задачи.

О примерах

В этой книге представлено много небольших фрагментов кода VBA, объясняющих отдельную тему или представляющих пример решения поставленной задачи. Зачастую этот код может состоять из единственного оператора. В некоторых случаях примеры представлены только *выражением*, которое не является корректной инструкцией.

Ниже представлено выражение.

```
Range("A1").Value
```

Чтобы протестировать выражение, его необходимо выполнить. Для этого часто используется функция `MsgBox`.

```
MsgBox Range("A1").Value
```

Чтобы выполнить предлагаемые примеры, поместите оператор в процедуре модуля VBA следующим образом.

```
Sub Test()
' Здесь вводится оператор
End Sub
```

Затем перенесите курсор в любое место процедуры и нажмите <F5>, чтобы выполнить ее. Кроме того, убедитесь, что код выполняется в правильном контексте. Например, если оператор ссылается на лист `Лист1`, удостоверьтесь, что в рабочей книге действительно есть лист с названием `Лист1`.

Код может представлять собой отдельный оператор. В таком случае для его тестирования используется окно Immediate. Окно Immediate применяется для немедленного выполнения операторов — без создания процедуры. Если окно Immediate не отображено на экране, то, работая в VBE, нажмите <Ctrl+G>.

Введите в окне Immediate оператор VBA и нажмите <Enter>. Чтобы проверить выражение в окне Immediate, введите перед ним знак вопроса (?). Знак вопроса — сокращенный символ команды `Print`. Например, вы можете ввести в окне Immediate следующее.

```
? Range("A1").Value
```

Результат выполнения выражения отображается в следующей строке окна Immediate.

Об объектах и коллекциях

Если вы полностью изучили первую часть этой главы, то должны иметь представление о VBA и знать основные методы управления модулями VBA в редакторе Visual Basic. Кроме того, вы ознакомились с примерами кода VBA и получили достаточно информации о таких элементах, как объекты и свойства. В этом разделе приводится дополнительная информация об объектах и коллекциях объектов.

Работая с кодом VBA, вы должны четко понимать назначение объектов и объектной модели Excel. Целесообразнее рассматривать объекты с точки зрения *иерархической структуры*. На вершине объектной модели находится объект `Application` — в данном случае, Excel. Но если вы программируете в VBA, запуская VBE в Microsoft Word, то объектом `Application` будет выступать Word.

Иерархия объектов

Объект `Application` (т.е. Excel) содержит другие объекты. Ниже приведено несколько примеров объектов, которые находятся в объекте `Application`.

- `Workbooks` (коллекция всех объектов `Workbook` — рабочих книг);
- `Windows` (коллекция всех объектов `Window` — окон);
- `AddIns` (коллекция всех объектов `AddIn` — надстроек).

Некоторые объекты могут содержать другие объекты. Например, коллекция `Workbooks` состоит из всех открытых объектов `Workbook`, а объект `Workbook` включает другие объекты, некоторые из которых представлены ниже.

- `Worksheets` (коллекция объектов `Worksheet` — рабочих листов);
- `Charts` (коллекция объектов `Chart` — диаграмм);
- `Names` (коллекция объектов `Name` — имен).

Каждый из этих объектов, в свою очередь, может содержать другие объекты. Коллекция `Worksheets` состоит из всех объектов `Worksheet` рабочей книги `Workbook`. Объект `Worksheet` включает другие объекты, среди которых следующие.

- `ChartObjects` (коллекция объектов `ChartObject` — элементов диаграмм);
- `Range` — диапазон;
- `PageSetup` — параметры страницы;
- `PivotTables` (коллекция объектов `PivotTable` — сводных таблиц).

Может быть, вы пока не готовы правильно воспринять подобную концепцию, но со временем наверняка поймете, что иерархия объектов вполне логична и хорошо структурирована. Кстати, вся объектная модель Excel схематически изображена в электронной справочной системе.

О коллекциях

Одной из ключевых концепций в программировании на языке VBA являются *коллекции*. Коллекция — это группа объектов одного класса (и сама коллекция тоже является объектом). Как указывалось выше, `Workbooks` — это коллекция всех открытых в данный момент объектов `Workbook`. `Worksheets` — коллекция всех объектов `Worksheet`, которые содержатся в конкретном объекте `Workbook`. Вы можете одновременно управлять целой коллекцией объектов или отдельным объектом этой коллекции. Чтобы сослаться на один объект из коллекции, введите название или номер объекта в скобках после названия коллекции.

```
Worksheets("Лист1")
```

Если лист `Лист1` — это первый рабочий лист в коллекции, то можно использовать следующую ссылку.

```
Worksheets(1)
```

На второй лист в рабочей книге `Workbook` ссылаются как на `Worksheets(2)` и т.д.

Кроме того, существует коллекция с названием `Sheets`, состоящая из всех листов рабочей книги: рабочих листов и листов диаграмм. Если `Лист1` — первый лист в книге, то на него можно сослаться так.

```
Sheets(1)
```

Ссылки на объекты

Если вы обращаетесь к объекту в VBA, то в ссылке на него вводятся названия всех расположенных выше в иерархической структуре объектов, разделенных точкой. Что делать, если в Excel открыты две рабочие книги, и в обеих имеется рабочий лист с названием `Лист1`? В этом случае в ссылке требуется указать *контейнер* требуемого объекта.

```
Workbooks("Книга1").Worksheets("Лист1")
```

Без указания рабочей книги редактор Visual Basic искал бы лист `Лист1` в активной рабочей книге.

Чтобы сослаться на определенный диапазон (например, ячейку `A1`) на рабочем листе с названием `Лист1` в рабочей книге `Книга1`, можно использовать следующее выражение.

```
Workbooks("Книга1").Worksheets("Лист1").Range("A1")
```

Полная ссылка из предыдущего примера включает объект `Application` и выглядит таким образом.

```
Application.Workbooks("Книга1").Worksheets("Лист1").Range("A1")
```

Однако в большинстве случаев можно опускать объект `Application` в ссылках (кроме него использоваться больше нечему). Если объект `Книга1` — это активная рабочая книга, то опустите ссылку на нее и запишите рассматриваемое выражение следующим образом.

```
Worksheets("Лист1").Range("A1")
```

Если `Лист1` является активным рабочим листом, можно еще более упростить выражение.

```
Range("A1")
```

НА ЗАМЕТКУ

В Excel отсутствует объект отдельной ячейки. Отдельная ячейка представляет собой объект `Range`, состоящий из одного элемента.

Простые ссылки на объекты (как в приведенных примерах) ничего не выполняют. Чтобы выполнить действие, прочтите или измените свойства объекта или задайте метод, который выполняется по отношению к объекту.

Свойства и методы

Запутаться в свойствах и методах несложно: их существует несколько тысяч. В этом разделе показано, как осуществляется доступ к свойствам и методам объектов.

Свойства объектов

Все объекты обладают свойствами. Например, объект `Range` обладает свойством с названием `Value`. Вы можете создать оператор VBA, чтобы отобразить свойство `Value` или задать свойству `Value` определенное значение. Ниже приведена процедура,

использующая функцию VBA `MsgBox` для отображения окна, в котором представлено значение ячейки `A1` листа `Лист1` активной рабочей книги.

```
Sub ShowValue()
     MsgBox Worksheets("Лист1").Range("A1").Value
End Sub
```

`MsgBox` — полезная функция, часто используемая для отображения результатов выполнения операторов VBA. В этой книге она применяется очень часто.

Код предыдущего примера отображает текущее значение свойства `Value` для конкретной ячейки — `A1` рабочего листа `Лист1` активной рабочей книги. Обратите внимание, что если в активной книге отсутствует лист с названием `Лист1`, то макрос выдаст ошибку.

Что необходимо сделать, чтобы изменить свойство `Value`? Ниже приведена процедура по изменению значения ячейки `A1` путем определения значения свойства `Value`.

```
Sub ChangeValue()
  Worksheets("Лист1").Range("A1").Value = 123
End Sub
```

После выполнения этой процедуры ячейка `A1` листа `Лист1` получает значение 123. Вы можете ввести описанные процедуры в модуль и протестировать их.

Многие объекты имеют свойство по умолчанию. Для объекта `Range` свойством по умолчанию является `Value`. Следовательно, выражение `.Value` в приведенном выше коде можно опустить, и ничего не изменится. Однако лучше включать ссылку на свойство, даже если оно используется по умолчанию.

Методы объектов

Кроме свойств, объекты характеризуются методами. *Метод* — это действие, которое выполняется над объектом. Ниже приведен простой пример использования метода `Clear` по отношению к диапазону ячеек. После выполнения этой процедуры ячейки `A1:C3` листа `Лист1` станут пустыми, и дополнительное их форматирование будет удалено.

```
Sub ZapRange()
     Worksheets("Лист1").Range("A1:C3").Clear
End Sub
```

Если необходимо удалить значения в диапазоне, но оставить форматирование, используйте метод `ClearContents` объекта `Range`.

Многие методы получают аргументы, определяющие выполняемые над объектом действия более детально. Далее приводится пример, в котором ячейка `A1` копируется в ячейку `B1` с помощью метода `Copy` объекта `Range`. В данном примере метод `Copy` получает один аргумент (адрес ячейки, в которую следует скопировать данные). Обратите внимание, что в примере используется символ продолжения строки (пробел и подчеркивание). Вы можете не применять этого символа, а ввести оператор в одну строку.

```
Sub CopyOne()
     Worksheets("Лист1").Range("A1").Copy _
          Worksheets("Лист1").Range("B1")
End Sub
```

Определение аргументов методов и свойств

В среде программистов VBA определение аргументов методов и свойств часто вызывает определенные трудности. Некоторые методы используют аргументы для дальнейшего уточнения действия; отдельные свойства используют аргументы для дальнейшего определения значения свойства. Иногда один или несколько аргументов вообще применять не обязательно.

Если метод использует аргументы, они указываются после названия метода и разделяются запятыми. Если метод использует необязательные аргументы, то можете пропустить их, оставив пустые места. Рассмотрим метод `Protect` объекта рабочей книги. В справочной системе дается информация о том, что метод `Protect` имеет три аргумента: пароль, структуру, окна. Эти аргументы соответствуют параметрам в диалоговом окне Защита книги.

К примеру, если требуется защитить рабочую книгу под названием `MyBook.xls`, используйте такой оператор.

```
Workbooks("MyBook.xls").Protect "xyzzy", True, False
```

В данном случае рабочая книга защищена паролем (аргумент 1). Также защищена структура рабочей книги (аргумент 2), но не ее окна (аргумент 3).

Если вы не хотите присваивать пароль, можно применить следующий оператор.

```
Workbooks("MyBook.xls").Protect , True, False
```

Обратите внимание, что первый аргумент пропущен, а его место обозначено с помощью запятой.

Существует и другой подход (причем в этом случае программу удобнее будет читать) — использование именованных аргументов. Применим именованные аргументы для предыдущего примера.

```
Workbooks("MyBook.xls").Protect , Structure:=True, Windows:=False
```

Применение именованных аргументов — хорошая идея, особенно в методах с большим количеством необязательных аргументов, когда следует использовать только некоторые из них. При работе с именованными аргументами не требуется оставлять место для пропущенных аргументов.

Для свойств, использующих аргументы, последние указываются в скобках. Например, свойство `Address` объекта `Range` имеет пять аргументов — все необязательные. Показанный ниже оператор некорректен, так как пропущены скобки.

```
MsgBox Range("A1").Address False          ' некорректно
```

Правильный синтаксис для этого оператора выглядит так.

```
MsgBox Range("A1").Address(False)
```

Кроме того, оператор может записываться с использованием именованного аргумента.

```
MsgBox Range("A1").Address(rowAbsolute:=False)
```

Подобные тонкости применения методов и свойств вскоре станут для вас привычным делом, вы даже не будете задумываться, почему это так.

Объект Comment: пример использования

Чтобы лучше разобраться со свойствами и методами объекта, сосредоточимся на изучении конкретного объекта — `Comment`. Объект `Comment` создается с помощью команды Excel Вставка⇨Примечание и предназначается для вставки комментария в ячейки. В следующих разделах вы получите более детальное представление об управлении этим объектом. Не волнуйтесь, если материал этого раздела покажется вам слишком обширным. Представленные концепции со временем станут значительно понятнее.

Справочные сведения по объекту Comment

Единственный способ получить информацию о конкретном объекте — найти соответствующие разделы в электронной справочной системе. На рис. 7.14 показано главное окно справочной системы для объекта `Comment`.

Обратите внимание, что подчеркнутые или выделенные другим цветом слова — это гиперссылки, отображающие дополнительную информацию. Например, можно щелкнуть на гиперссылке Properties, чтобы отобразить список всех свойств объекта `Comment`, или на ссылке Methods, чтобы представить список всех методов объекта.

Использование электронной справочной системы

Самый простой способ получить справку о конкретном объекте, свойстве или методе — ввести ключевое слово в окне кода и нажать <F1>. Если это ключевое слово трактуется неоднозначно, появляется диалоговое окно выбора темы.

К сожалению, элементы, перечисленные в этом диалоговом окне, не всегда понятны, поэтому, чтобы найти нужный раздел, часто приходится обращаться к методу проб и ошибок. Показанное на рисунке диалоговое окно представлено для случая введения **Comment** и нажатия <F1>. В рассматриваемом примере `Comment` является объектом, однако он может также вести себя как свойство. При щелчке на первой теме отображается раздел, посвященный объекту `Comment`; если вы щелкнете на второй теме, то увидите раздел для свойства `Comment`.

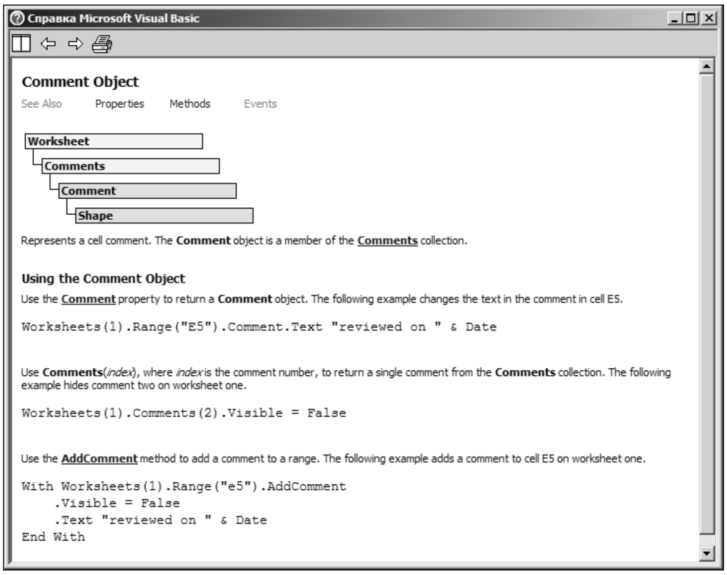


Рис. 7.14. Главное окно справочной системы для объекта `Comment`

Свойства объекта Comment

Объект `Comment` обладает шестью свойствами. В табл. 7.1 вы найдете полный их список и краткое описание каждого свойства. Если свойство *доступно только для чтения*, это значит, что программа VBA может только получать свойство, но не изменять его.

Таблица 7.1. Свойства объекта Comment

Свойство	Только для чтения	Описание
`Application`	Да	Возвращает объект, представляющий приложение, в котором создавалось примечание (т.е. Excel)
`Author`	Да	Возвращает имя человека, создавшего примечание
`Creator`	Да	Возвращает число, указывающее приложение, в котором создавался объект. Не используется в Excel для Windows (применяется только в Excel для Macintosh)
`Parent`	Да	Возвращает родительский объект для примечания (это всегда объект `Range`)
`Shape`	Да	Возвращает объект `Shape`, который представляет форму, присоединенную к примечанию
`Visible`	Нет	Если это свойство имеет значение `True`, то примечание отображается на экране

Методы объекта Comment

В табл. 7.2 приведены методы, которые можно использовать в объекте `Comment`. Все они выполняют обычные операции, которые производятся над примечанием вручную... Однако вы никогда прежде не рассматривали эти действия как методы.

Таблица 7.2. Методы объекта Comment

Метод	Описание
`Delete`	Удаляет комментарий
`Next`	Возвращает объект `Comment`, представляющий следующий комментарий
`Previous`	Возвращает объект `Comment`, представляющий предыдущий комментарий
`Text`	Возвращает или определяет текст в комментарии (метод имеет три аргумента)

Возможно, вас удивило, что `Text` — это метод, а не свойство. Этот формат приводит нас к важному умозаключению: различия между свойствами и методами не всегда четкие, а объектная модель не является идеально последовательной. На самом деле неважно, насколько точно вы будете различать свойства и методы. Пока используется правильный синтаксис, не имеет значения, какую роль в коде выполняет ключевое слово — свойства или метода.

Коллекция Comments

Как вы помните, коллекция — это группа одинаковых объектов. Каждый рабочий лист имеет коллекцию `Comments`, состоящую из всех объектов `Comment` рабочего листа. Если на рабочем листе отсутствуют примечания, эта коллекция пуста.

Например, приведенный далее код ссылается на первое примечание листа `Лист1` активной рабочей книги.

```
Worksheets("Лист1").Comments(1)
```

Следующий оператор отображает текст, который содержится в первом примечании листа Лист1.

```
MsgBox Worksheets("Лист1").Comments(1).Text
```

В отличие от большинства объектов, объект Comment не имеет свойства Name. Следовательно, чтобы сослаться на конкретный комментарий, используйте номер, а для получения необходимого комментария обратитесь к свойству Comment объекта Range (см. далее).

Коллекция Comments — тоже объект, имеющий собственный набор свойств и методов. Например, следующий пример определяет общее количество комментариев.

```
MsgBox ActiveSheet.Comments.Count
```

В данном случае используется свойство Count коллекции Comments, в котором хранится количество объектов Comment на активном рабочем листе. В следующем примере показан адрес ячейки, содержащей первое примечание.

```
MsgBox ActiveSheet.Comments(1).Parent.Address
```

В этом примере Comments(1) возвращает первый объект Comment коллекции Comments. Свойство Parent объекта Comment возвращает его контейнер, представленный объектом Range. В окне сообщений отображается свойство Address объекта Range. В итоге оператор показывает адрес ячейки, содержащей первое примечание.

Кроме того, вы можете циклически просмотреть все примечания на листе, используя конструкцию For Each-Next (о ней речь пойдет в главе 8). Ниже приведен пример использования отдельных сообщений для раздельного отображения каждого примечания активного рабочего листа.

```
For Each cmt in ActiveSheet.Comments
  MsgBox cmt.Text
Next cmt
```

Если вы не хотите, чтобы на экране находилось большое количество диалоговых окон с сообщениями, то используйте следующую процедуру для вывода всех примечаний в одном окне Intermediate.

```
For Each cmt in ActiveSheet.Comments
  Debug.Print cmt.Text
Next cmt
```

О свойстве Comment

В этом разделе речь идет об объекте Comment. В справочной системе указано, что объект Range обладает свойством Comment. Если ячейка содержит примечание, свойство Comment возвращает объект — объект Comment. Например, следующий оператор ссылается на объект Comment ячейки A1.

```
Range("A1").Comment
```

Если это первое примечание на листе, то на данный объект Comment можно сослаться следующим образом.

```
Comments(1)
```

Чтобы отобразить примечание ячейки A1 в окне сообщения, используйте такой оператор.

```
MsgBox Range("A1").Comment.Text
```

Если в ячейке A1 нет примечания, то оператор выдаст ошибку.

Тот факт, что свойство может возвращать объект, довольно важен (возможно, это сложно понять, но данная концепция имеет решающее значение в программировании на VBA).

Объекты, вложенные в Comment

Управление свойствами сначала кажется сложной задачей, потому что некоторые из них возвращают объекты. Предположим, необходимо определить цвет фона конкретного примечания на листе Лист1. Просмотрев список свойств объекта Comment, вы не найдете ничего, что относится к определению цвета. Выполните следующие действия.

1. Используйте свойство Shape объекта Comment, возвращающее объект Shape, который содержится в примечании.
2. Используйте свойство Fill объекта Shape, возвращающее объект FillFormat.
3. Используйте свойство ForeColor объекта FillFormat, возвращающее объект ColorFormat.
4. Используйте свойство RGB (или свойство SchemeColor) объекта ColorFormat, чтобы задать цвет.

Иначе говоря, получение цвета фона объекта Comment связано с доступом к другим объектам, которые в нем содержатся. Ниже описана иерархия задействованных объектов.

```
Application (Excel)
  Workbook
    Worksheet
      Comment
        Shape
          FillFormat
            ColorFormat
```

Следует предупредить, что в этом можно легко запутаться! Но в качестве примера "элегантности" VBA посмотрите, как код для изменения цвета примечания можно записать с помощью одного оператора.

```
Worksheets("Лист1").Comments(1).Shape.Fill.ForeColor _
     .RGB = RGB(0, 255, 0)
```

Вы вправе использовать также свойство SchemeColor (задаваемое в диапазоне от 0 до 80).

```
Worksheets("Лист1").Comments(1).Shape.Fill.ForeColor _
     .SchemeColor = 12
```

В данном типе ссылки сразу трудно разобраться, но впоследствии вам несложно будет ориентироваться в иерархии объектов. Вы быстро изучите ее особенности, прежде всего, потому, что в Excel при записи последовательности действий практически всегда вопрос иерархии задействованных объектов ставится на первое место.

Смущают цвета?

Когда вы получите определенный опыт в использовании VBA и приступите к указанию цветов для различных объектов, то почти наверняка столкнетесь с трудностями. Помните: Excel применяет 56-цветную палитру цветов, причем все цвета сохраняются в каждой рабочей книге. Это те цвета, которые отображаются при использовании кнопки Цвет заливки на панели инструментов Форматирование (данные цвета отображаются на вкладке Цвет диалогового окна Параметры, вызываемого из меню Сервис). Что это означает для программиста VBA? Цвет, который вы задаете в коде VBA, не всегда соответствует тому, который появляется на экране.

Ситуация может усложниться еще больше. В зависимости от объекта, с которым вы работаете, для задания цвета используются различные объекты и свойства.

Цвет объекта `Shape` можно задать с помощью свойства `RGB` или свойства `SchemeColor`. Свойство `RGB` позволяет определить цвет в виде значений красного, зеленого и синего компонентов. Это свойство аналогично функции `RGB`, имеющей три аргумента, каждый из которых задается в диапазоне от 0 до 255. Функция `RGB` возвращает значение в диапазоне от 0 до 16777215. Но Excel, как уже упоминалось, может обрабатывать только 56 цветов. Поэтому фактический цвет, полученный при использовании функции `RGB`, будет самым точным соответствием заданному цвету из 56-цветовой палитры рабочей книги. Свойство `SchemeColor` принимает значения от 0 до 80. В справочной системе вы не найдете ничего о том, что в действительности представляют собой эти цвета. Однако они ограничены образцами цветов на палитре рабочей книги.

При работе с цветами в объекте `Range` вам придется обратиться к его вложенному объекту `Interior`. Вы можете задать цвет с помощью одного из свойств последнего: `Color` или `ColorIndex`. Корректные значения свойства `ColorIndex` находятся в диапазоне от 0 до 56 (0 означает отсутствие заливки). Эти значения соответствуют палитре цветов рабочей книги. К сожалению, порядок, в котором отображаются цвета, совершенно не связан с системой нумерации значений свойства `ColorIndex`, поэтому для определения с помощью `ColorIndex` конкретного цвета лучше записать макрос. Однако даже в этом случае не будет гарантии, что пользователь не изменил цветовую палитру рабочей книги. В последнем случае свойство `ColorIndex` выдаст далеко не тот результат, который вы ожидали.

При использовании свойства `Color` можно определить значение цвета с помощью функции `RGB`. Однако помните, что фактически отображаемый цвет будет всего лишь ближайшим цветом на палитре рабочей книги, который соответствует заданному вами значению.

Кстати, чтобы изменить цвет текста в примечании, обратитесь к объекту `TextFrame` объекта `Comment`, который содержит объект `Characters`, включающий, в свою очередь, объект `Font`. Далее обратитесь к свойствам `Color` или `ColorIndex` объекта `Font`. Ниже приведен пример, устанавливающий свойство `ColorIndex` в значение 5.

```
Worksheets("Лист1").Comments(1) _
     .Shape.TextFrame.Characters.Font.ColorIndex = 5
```

Содержит ли ячейка примечание?

Следующий оператор отображает примечание ячейки `A1` активного листа.

```
MsgBox Range("A1").Comment.Text
```

Если в ячейке `A1` примечание отсутствует, при выполнении этого оператора возникнет непонятное сообщение об ошибке: `Object variable or With block variable not set`.

Чтобы определить, содержит ли конкретная ячейка примечание, напишите код, проверяющий, не пустой ли объект `Comment`, — т.е. равен ли он `Nothing` (это кор-

ректное ключевое слово VBA). Следующий оператор отображает `True`, если в ячейке `A1` примечание отсутствует.

```
MsgBox Range("A1").Comment Is Nothing
```

Обратите внимание, что в этом примере используется ключевое слово `Is`, а не знак равенства.

Добавление нового объекта Comment

Возможно, вы заметили, что в списке методов объекта `Comment` не существует метода для добавления нового примечания. Это объясняется тем, что метод `AddComment` принадлежит объекту `Range`. Следующий оператор добавляет примечание (пустое) в ячейку `A1` активного рабочего листа.

```
Range("A1").AddComment
```

Обратившись к справочной системе, вы обнаружите, что метод `AddComment` имеет аргумент, представляющий текст примечания. Следовательно, можно добавить примечание и текст в нем с помощью всего одного оператора.

```
Range("A1").AddComment "Формула разработана JW"
```

НА ЗАМЕТКУ

Метод `AddComment` генерирует ошибку, если ячейка уже содержит примечание.

НА КОМПАКТ-ДИСКЕ

Если вы хотите увидеть рассмотренные свойства и методы объекта `Comment` в действии, посмотрите пример на прилагаемом к книге компакт-диске. Соответствующая рабочая книга содержит несколько примеров управления объектами `Comment` с помощью кода VBA. Скорее всего, вы не поймете весь код, но на данном этапе осознаете, как можно использовать VBA для управления объектом.

Полезные свойства объекта Application

Как вы знаете, при работе в Excel активной одновременно может быть только одна рабочая книга. И если вы управляете рабочим листом, то активна на нем только одна ячейка (даже если выделен диапазон).

VBA это известно, поэтому вы можете ссылаться на активные объекты более простым методом. Это удобно, так как вы не всегда знаете, с какой именно рабочей книгой, рабочим листом или ячейкой будете работать. VBA представляет свойства объекта `Application` для определения этого. Например, объект `Application` обладает свойством `ActiveCell`, возвращающим ссылку на активную ячейку. Следующая инструкция присваивает значение 1 активной ячейке.

```
ActiveCell.Value = 1
```

Обратите внимание, что в этом примере пропущена ссылка на объект `Application`, так как это само собой разумеется. Важно понять, что такая инструкция может выдать ошибку, если активный лист не является рабочим. Например, если VBA выполняет этот оператор, когда активен лист диаграммы, то процедура прекращает выполняться, а на экране отображается сообщение об ошибке.

Если на рабочем листе выделен диапазон ячеек, то активная ячейка будет находиться в выделенном диапазоне. Другими словами, активная ячейка всегда одна (их никогда не бывает несколько).

Объект `Application` также обладает свойством `Selection`, возвращающим ссылку на выделенный объект, т.е. отдельную ячейку (активную), диапазон ячеек или объект типа `ChartObject`, `TextBox`, `Shape`.

В табл. 7.3 перечислены свойства объекта `Application`, которые вам пригодятся при работе с ячейками и диапазонами ячеек.

Таблица 7.3. Некоторые полезные свойства объекта Application

Свойство	Возвращаемый объект
`ActiveCell`	Активная ячейка
`ActiveChart`	Активный лист диаграммы или объект диаграммы на рабочем листе. Если диаграмма не активна, то свойство равно `Nothing`
`ActiveSheet`	Активный лист (рабочий лист или лист диаграммы)
`ActiveWindow`	Активное окно
`ActiveWorkbook`	Активная рабочая книга
`RangeSelection`	Выделенные ячейки на рабочем листе в заданном окне, даже если выделен графический объект (на самом деле это свойство объекта `Window`)
`Selection`	Выделенный объект (объект `Range`, `Shape`, `ChartObject` и т.д.)
`ThisWorkbook`	Рабочая книга, содержащая выполняемую процедуру

Преимущество использования этих свойств для получения объекта заключается в том, что совершенно не обязательно знать, какая ячейка, рабочий лист или книга являются активными, и вводить конкретную ссылку на этот объект. Данный факт позволяет создавать код VBA, который не ограничивается конкретной книгой, листом или ячейкой. Например, следующая инструкция удаляет содержимое активной ячейки, даже если адрес активной ячейки не известен.

```
ActiveCell.ClearContents
```

В следующем примере отображается сообщение, указывающее имя активного листа.

```
MsgBox ActiveSheet.Name
```

Если требуется узнать название активной рабочей книги, используйте такой оператор.

```
MsgBox ActiveBook.Name
```

Если на рабочем листе выделен диапазон, то заполните последний одним значением, выполнив единственный оператор. В следующем примере свойство `Selection` объекта `Application` возвращает объект `Range`, соответствующий выделенным ячейкам. Оператор изменяет свойство `Value` этого объекта `Range`, и в результате получается диапазон, заполненный одним значением.

```
Selection.Value = 12
```

Обратите внимание: если выделен не диапазон ячеек (например, объект `ChartObject` или `Shape`), то этот оператор выдаст ошибку, так как объекты `ChartObject` и `Shape` не обладают свойством `Value`.

Однако приведенный ниже оператор присваивает объекту `Range`, который выделялся перед выделением другого объекта (отличного от диапазона ячеек), значение 12. В справочной системе указано, что свойство `RangeSelection` относится только к объекту `Window`.

```
ActiveWindow.RangeSelection.Value = 12
```

Чтобы узнать, сколько ячеек выделено на рабочем листе, применяется свойство `Count`.

```
MsgBox ActiveWindow.RangeSelection.Count
```

Работа с объектами Range

В основном, работа, которая выполняется в VBA, связана с управлением ячейками и диапазонами на рабочих листах, что и является основным предназначением электронных таблиц. Вопрос управления ячейками в VBA рассматривался ранее при обсуждении относительного и абсолютного способов записи макросов, однако вам необходимо уделить ему особое внимание.

Объект `Range` содержится в объекте `Worksheet` и состоит из одной ячейки или диапазона ячеек на отдельном рабочем листе. В следующих разделах будут рассмотрены три способа задания ссылки на объекты `Range` в программе VBA.

- Свойство `Range` объекта класса `Worksheet` или `Range`.
- Свойство `Cells` объекта `Worksheet`.
- Свойство `Offset` объекта `Range`.

Свойство Range

Свойство `Range` возвращает объект `Range`. Из справочных сведений по свойству `Range` вы узнаете, что к нему можно обратиться с помощью нескольких вариантов синтаксиса.

```
объект.Range(ячейка1);
объект.Range(ячейка1, ячейка2).
```

Свойство `Range` относится к одному из двух типов объектов: объекту `Worksheet` или объекту `Range`. В данном случае *`ячейка1`* и *`ячейка2`* указывают параметры, которые Excel будет воспринимать как идентифицирующие диапазон (в первом случае) или ***очерчивающие*** диапазон (во втором случае). Ниже следует несколько примеров использования метода `Range`.

В данной главе уже рассматривались примеры, подобные представленным ниже. Далее приведена инструкция, которая вводит значение в указанную ячейку: значение 1 вводится в ячейку `A1` на листе `Лист1` активной рабочей книги.

```
Worksheets("Лист1").Range("A1").Value = 1
```

Свойство `Range` также поддерживает имена, определенные в рабочих книгах. Поэтому если ячейка называется Ввод, то для ввода значения в нее может использоваться следующий оператор.

```
Worksheets("Лист1").Range("Ввод").Value = 1
```

В следующем примере в диапазон из 20-ти ячеек на активном листе вводится одинаковое значение. Если активный лист не является рабочим листом, то отображается сообщение об ошибке.

```
ActiveSheet.Range("A1:B10").Value = 2
```

Приведенный ниже пример приведет к тому же результату, что и предыдущий.

```
Range("A1", "B10") = 2
```

Отличие заключается лишь в том, что опущена ссылка на лист, поэтому предполагается активный рабочий лист. Кроме того, пропущено свойство, поэтому используется свойство по умолчанию (для объекта Range это свойство Value). В приведенном примере используется второй синтаксис ссылки на свойство Range. В данном случае первый аргумент — это левая верхняя ячейка диапазона, а второй аргумент — это ячейка в правом нижнем углу диапазона.

В следующем примере для получения пересечения двух диапазонов применяется оператор пересечения Excel (пробел). Пересечением является одна ячейка — C6. Следовательно, данный оператор вводит значение 3 в ячейку C6.

```
Range("C1:C10 A6:E6") = 3
```

Наконец, в следующем примере значение 4 вводится в пять ячеек, т.е. в независимые диапазоны. Запятая выполняет роль оператора объединения.

```
Range("A1,A3,A5,A7,A9") = 4
```

До настоящего момента во всех рассмотренных примерах использовалось свойство Range объекта Worksheet. Вы также можете использовать свойство Range объекта Range. Сначала будет непросто, однако постарайтесь разобраться.

Ниже показан пример использования свойства Range объекта Range (в данном случае объектом Range является активная ячейка). В этом примере объект Range рассматривается как левая верхняя ячейка на рабочем листе. В ячейку, которая в таком случае *была бы* B2, вводится значение 5. Другими словами, полученная ссылка является относительной для верхнего левого угла объекта Range. Следовательно, следующий оператор вводит значение 5 в ячейку, расположенную справа внизу от активной ячейки.

```
ActiveCell.Range("B2") = 5
```

Существует также намного более понятный способ обратиться к ячейке по отношению к диапазону — это свойство Offset (см. далее в этой главе).

Свойство Cells

Другим способом сослаться на диапазон является использование свойства Cells. Подобно Range, свойство Cells может использоваться в объектах Worksheet и Range. Справочная система указывает на три варианта синтаксиса свойства Cells.

```
объект.Cells(номер_строки, номер_столбца);
объект.Cells(номер_строки);
объект.Cells.
```

Проиллюстрируем на примерах особенности применения свойства Cells. Вначале в ячейку A1 листа Лист1 введите значение 9. В данном случае используется первый синтаксис, где аргументами являются номер строки (от 1 до 65536) и номер столбца (от 1 до 256).

```
Worksheets("Лист1").Cells(1, 1) = 9
```

Ниже приведен пример, в котором значение 7 вводится в ячейку D3 (т.е. пересечение строки 3, столбца 4) активного рабочего листа.

```
ActiveSheet.Cells(3, 4) = 7
```

Вы можете также использовать свойство Cells объекта Range. При этом объект Range, который возвращается свойством Cells, задается относительно левой верхней ячейки диапазона Range, на который мы ссылаемся. Сложно? Может быть. Приведем

пример. Следующая инструкция вводит значение 5 в активную ячейку. Помните, что в данном случае активная ячейка рассматривается как ячейка `A1` на рабочем листе.

```
ActiveCell.Cells(1, 1) = 5
```

Настоящее преимущество представленного типа ссылок на ячейки станет очевидным, когда речь пойдет о переменных и циклах (см. главу 8). В большинстве случаев в аргументах не будет использоваться фактическое значение. Вместо него применяется переменная.

Чтобы ввести значение 5 в ячейку, которая находится под активной, обратитесь к такой инструкции.

```
ActiveCell.Cells(2, 1) = 5
```

Предыдущий пример можно описать так: необходимо начать с активной ячейки, рассматривая ее как ячейку `A1`. Затем обратитесь к ячейке во второй строке и первом столбце диапазона.

Еще один синтаксис метода `Cells` использует один аргумент, который задается в диапазоне от 1 до 16777216. Второе число равно количеству ячеек на рабочем листе (65536 строк умножить на 256 столбцов). Ячейки нумеруются, начиная с `A1` вправо, затем вниз и вправо вдоль следующей строки. 256-я ячейка — это `IV1`, а 257-я — `A2`.

Далее в ячейку `H3` активного листа (520-ю ячейку на рабочем листе) введем значение 2.

```
ActiveSheet.Cells(520) = 2
```

Чтобы отобразить значение в последней ячейке на рабочем листе (`IV65536`), используйте оператор

```
MsgBox ActiveSheet.Cells(16777216)
```

Этот синтаксис можно использовать и с объектом `Range`. В таком случае будет получена ячейка по отношению к указанному объекту `Range`. Например, если объект `Range` — это диапазон `A1:D10` (40 ячеек), то свойство `Cells` может иметь аргумент от 1 до 40 и возвращать одну из ячеек объекта `Range`. В следующем примере значение 2000 вводится в ячейку `A2`, так как `A2` является пятой ячейкой (считая сверху направо, затем вниз) в указанном диапазоне.

```
Range("A1:D10").Cells(5) = 2000
```

В предыдущем примере аргумент свойства `Cells` не ограничен значениями между 1 и 40. Если аргумент превышает количество ячеек в диапазоне, счет продолжается, будто диапазон больше, чем он есть на самом деле. Следовательно, оператор, подобный предыдущему, может изменить значение ячейки, которая находится за пределами указанного диапазона `A1:D10`.

Третий синтаксис свойства `Cells` возвращает все ячейки на указанном рабочем листе. В отличие от двух других, в этом синтаксисе возвращаемыми в результате данными будет не одна ячейка, а целый диапазон. В приведенном ниже примере использован метод `ClearContents` по отношению к диапазону, полученному с помощью свойства `Cells` для активного рабочего листа. В результате будет удалено содержимое каждой ячейки на рабочем листе.

```
ActiveSheet.Cells.ClearContents
```

Свойство Offset

Свойство `Offset` (подобно свойствам `Range` и `Cells`) также возвращает объект `Range`. В отличие от рассмотренных выше свойств, `Offset` применяется только к объекту `Range` и ни к какому другому. Данное свойство использует единственный синтаксис.

объект.Offset(*сдвиг_строки*, *сдвиг_столбца*)

Два аргумента свойства `Offset` соответствуют смещению относительно левой верхней ячейки указанного диапазона `Range`. Эти аргументы могут быть положительными (сдвиг вниз или вправо), отрицательными (вверх или влево) или нулевыми. В приведенном ниже примере значение 12 вводится в ячейку, которая находится под активной.

```
ActiveCell.Offset(1,0).Value = 12
```

В следующем примере значение 15 вводится в ячейку над активной ячейкой.

```
ActiveCell.Offset(-1,0).Value = 15
```

Если активная ячейка находится в строке 1, то свойство `Offset` в предыдущем примере выдаст ошибку, так как оно возвращает несуществующий объект `Range`.

Свойство `Offset` особо эффективно при использовании переменных в цикле (см. следующую главу).

В процессе записи макроса в относительном режиме указания ссылки Excel использует свойство `Offset` для обращения к ячейкам относительно начальной позиции (т.е. активной в момент начала записи макроса ячейки). Например, для генерации следующего кода применена функция записи макросов. Вначале включим запись макроса (при активной ячейке `B1`), потом введем значение в ячейки `B1:B3`, а затем вновь вернемся к ячейке `B1`.

```
Sub Macro1()
     ActiveCell.FormulaR1C1 = "1"
     ActiveCell.Offset(1, 0).Range("A1").Select
     ActiveCell.FormulaR1C1 = "2"
     ActiveCell.Offset(1, 0).Range("A1").Select
     ActiveCell.FormulaR1C1 = "3"
     ActiveCell.Offset(-2, 0).Range("A1").Select
End Sub
```

При записи макросов используется свойство `FormulaR1C1`. Как правило, для ввода значения в ячейку применяется свойство `Value`. Однако при использовании `FormulaR1C1` или `Formula` результат будет таким же.

Также обратите внимание на то, что полученный код ссылается на ячейку `A1` — это довольно странно, так как данная ячейка даже не была задействована в макросе. Такая особенность процедуры записи макросов делает программу даже более сложной, чем необходимо. Вы можете удалить все ссылки на `Range("A1")`, но макрос все равно будет работать нормально.

```
Sub Modified Macro1()
     ActiveCell.FormulaR1C1 = "1"
     ActiveCell.Offset(1, 0).Select
     ActiveCell.FormulaR1C1 = "2"
     ActiveCell.Offset(1, 0).Select
     ActiveCell.FormulaR1C1 = "3"
     ActiveCell.Offset(-2, 0).Select
End Sub
```

Вы можете получить еще более эффективную версию макроса (например ту, которую я написал вручную), где вообще не выполняется выделение ячеек.

```
Sub Macro1()
     ActiveCell = 1
     ActiveCell.Offset(1, 0) = 2
     ActiveCell.Offset(-2, 0) = 3
End Sub
```

Что следует знать об объектах

В предыдущих разделах были изложены основы использования объектов (включая коллекции), свойств и методов. Однако при этом мы только затронули вершину айсберга.

Более сложные, но важные концепции

В этом разделе рассматриваются дополнительные концепции, которые незаменимы, если вы собираетесь стать профессиональным программистом на VBA. Эти концепции станут более понятными по мере приобретения опыта работы с VBA, а также после прочтения следующих глав книги.

- Объекты обладают уникальными свойствами и методами.

  Каждый объект имеет собственный набор свойств и методов. Однако некоторые объекты характеризуются общими свойствами (например, `Name`) и методами (например, `Delete`).

- Для управления объектами их не обязательно выделять.

  Это может противоречить обычному представлению об управлении объектами в Excel, особенно если вы занимались программированием макросов XLM. Дело в том, что действия над объектами эффективнее выполнять, если их сначала не выделять. При записи макроса в Excel объект обычно сначала выделяется. Это не обязательно, хотя существенно замедляет работу макроса.

- Важно понимать предназначение коллекций.

  В большинстве случаев вы будете ссылаться на объект непосредственно, обращаясь к коллекции, к которой он принадлежит. Например, для обращения к объекту `Workbook` с названием `Myfile` необходимо сослаться на коллекцию `Workbooks` следующим образом.

  ```
  Workbooks("Myfile.xls")
  ```

  Эта ссылка возвращает объект — рабочую книгу, которая вас интересует.

- Свойства могут возвращать ссылку на другой объект. Например, в следующем операторе свойство `Font` возвращает объект `Font`, который содержится в объекте `Range`.

  ```
  Range("A1").Font.Bold = True
  ```

- Чтобы обратиться к одному и тому же объекту, можно воспользоваться несколькими способами.

  Предположим, что у вас есть рабочая книга с названием `Sales`, и это единственная открытая рабочая книга. В ней находится один лист с названием `Summary`. Вы можете сослаться на этот лист любым из приведенных ниже способов.

```
Workbooks("Sales.xls").Worksheets("Summary")
Workbooks(1).Worksheets(1)
Workbooks(1).Sheets(1)
Application.ActiveWorkbook.ActiveSheet
ActiveWorkbook.ActiveSheet
ActiveSheet
```

Используемый вами метод обычно зависит от того, насколько хорошо вам известно рабочее пространство. Например, если открыто несколько рабочих книг, то второй или третий методы не подойдут. Если вы будете работать с активным листом (какой бы это ни был лист), то воспользуйтесь последними тремя методами. Абсолютную уверенность в том, что вы ссылаетесь на конкретный лист в конкретной рабочей книге, дает только первый метод.

Узнайте больше об объектах и свойствах

Если это ваше первое знакомство с VBA, то вас могут несколько смутить многочисленные объекты, свойства и методы. Вы можете попытаться получить доступ к свойству, которым объект не обладает, однако в процессе выполнения кода произойдет ошибка. Он будет прерываться в этом месте до тех пор, пока вы не исправите ошибку.

К счастью, существует несколько способов узнать больше информации об объектах, свойствах и методах.

ПРОЧТИТЕ ОСТАВШУЮСЯ ЧАСТЬ КНИГИ

Не забывайте, что эта глава называется “Введение в Visual Basic for Applications”. В остальных главах книги рассмотрены детали, приведены полезные информативные примеры.

ИСПОЛЬЗУЙТЕ ФУНКЦИЮ ЗАПИСИ МАКРОСОВ

Несомненно, лучший способ ознакомиться с VBA — включить функцию записи макросов и записать отдельные действия, выполненные в Excel. Это быстрый метод узнать, какие объекты, свойства и методы относятся к конкретной задаче. Будет лучше, если при записи отображается окно модуля VBA, в котором представлен записываемый код.

ИСПОЛЬЗУЙТЕ ЭЛЕКТРОННУЮ СПРАВОЧНУЮ СИСТЕМУ

Основной источник подробной информации об объектах, методах и процедурах Excel — это электронная справочная система.

На рис. 7.15 демонстрируется раздел справочной системы по свойству `Value`. Это свойство относится к ряду объектов, и раздел справки содержит гиперссылки See Also (См. также), Applies To (Применимо к) и Example (Пример). Если вы щелкнете на ссылке See Also, то получите список связанных с данным ключевым словом тем (если такие темы существуют). При щелчке на ссылке Applies To отображается окно, где перечислены все объекты, использующие это свойство. Если вы щелкнете на ссылке Example, то получите возможность просмотреть один или несколько примеров (текст примера можно скопировать и вставить в модуль VBA, чтобы протестировать его на практике).

ИСПОЛЬЗУЙТЕ БРАУЗЕР ОБЪЕКТОВ

Окно Object Browser (Браузер объектов) — это удобный инструмент, предоставляющий список всех свойств и методов для всех доступных объектов. В VBE окно Object Browser можно отобразить одним из трех способов.

- ♦ Нажать <F2>.
- ♦ Выбрать в строке меню команду View⇨Object Browser.
- ♦ Щелкнуть на кнопке Object Browser на стандартной панели инструментов.

Окно Object Browser показано на рис. 7.16.

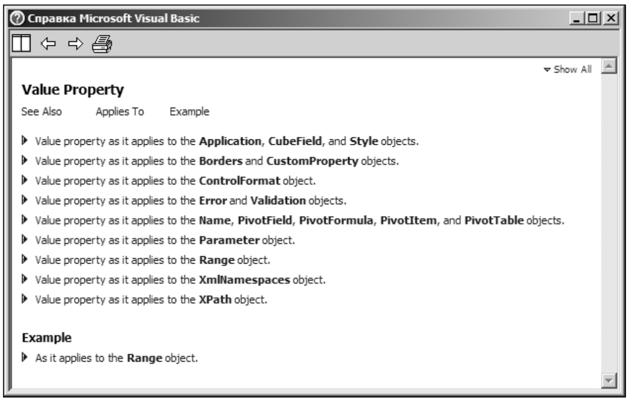


Рис. 7.15. Типичное окно справочной системы VBE

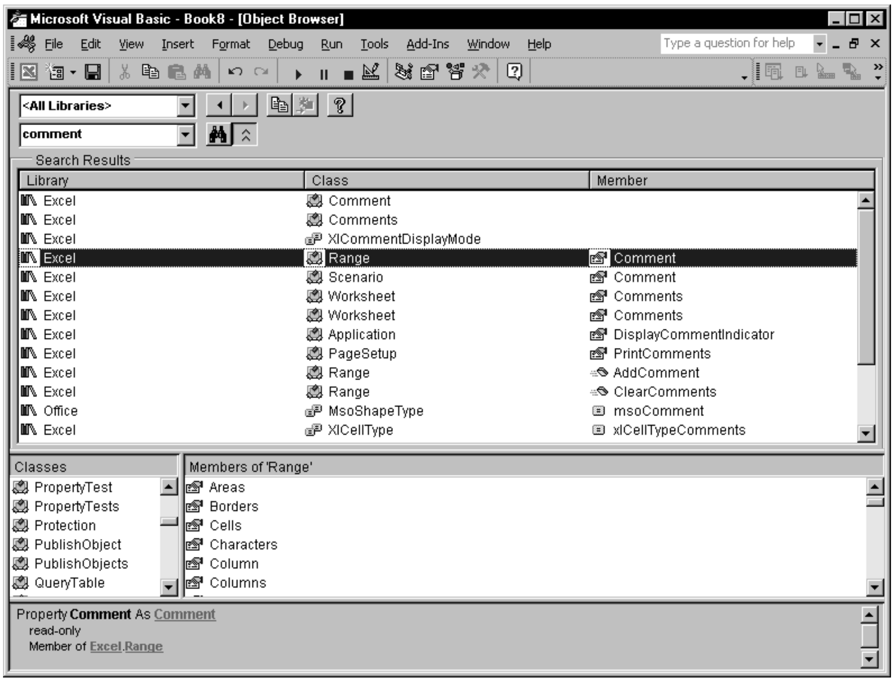


Рис. 7.16. Браузер объектов — полный справочный ресурс

Раскрывающийся список в левом верхнем углу окна Object Browser содержит список всех библиотек объектов, к которым у вас есть доступ.

- Собственно Excel.
- `MSForms` (используется для создания специальных диалоговых окон).
- `Office` (объекты, общие для всех приложений Microsoft Office).
- `Stdole` (объекты автоматизации OLE).
- `VBA`.
- Все открытые рабочие книги (каждая книга считается библиотекой объектов, так как содержит объекты).

Ваш выбор в этом списке определяет, что отображается в разделе Classes (Классы), а выбор в разделе Classes обусловит появление определенных компонентов в поле Members of (Включены в).

После выбора библиотеки можно осуществить поиск конкретной строки текста, чтобы получить список свойств и методов, содержащих данный текст. Это можно сделать, введя текст во втором раскрывающемся списке и щелкнув на значке с изображением бинокля. Предположим, что вы работаете над проектом, обрабатывающим примечания в ячейках.

1. Выберите интересующую вас библиотеку. Если вы не уверены, какую именно библиотеку выбрать, укажите вариант <All Libraries>.
2. Введите **Comment** в раскрывающемся списке под списком библиотек.
3. Щелкните на значке в виде бинокля, чтобы начать поиск текста.

В области Search Results (Результаты поиска) отображается текст, соответствующий фрагменту для поиска. Выберите один объект, чтобы отобразить его классы в разделе Classes. Укажите класс, чтобы отобразить его члены (свойства, методы и константы). Обратите внимание на нижнюю часть окна, где приведена дополнительная информация об объекте. Вы можете нажать <F1>, чтобы перейти непосредственно к необходимому разделу справочной системы.

Структура окна Object Browser может сначала показаться сложной, но, изучив ее, вы убедитесь в ее незаменимости.

ЭКСПЕРИМЕНТИРУЙТЕ С ОКНОМ IMMEDIATE

Как было отмечено во врезке в одном из предыдущих разделов этой главы, окно Immediate в VBE используется для тестирования операторов и проверки разных выражений VBA. Рекомендуем всегда отображать окно Immediate на экране, так как оно часто используется для проверки выражений и при отладке кода.

Глава 8

Основы программирования на VBA

В ЭТОЙ ГЛАВЕ...

В предыдущей главе вы ознакомились с основами VBA, теперь настало время изучить более сложные понятия. В этой главе рассматриваются ключевые элементы языка и концепции программирования на VBA.

- Элементы языка VBA: переменные, типы данных, константы и массивы
- Использование встроенных функций VBA
- Работа с объектами и коллекциями
- Контроль за выполнением процедур

Если вы ранее использовали другие языки программирования, то большая часть приведенной далее информации покажется вам знакомой. Однако в VBA есть свои тонкости, поэтому даже опытные программисты найдут в данной главе полезную информацию.

Элементы языка VBA. Обзор

В главе 7 речь шла об объектах, свойствах и методах. Но полученных вами знаний еще не достаточно для выполнения сложных задач. Данная глава введет вас в курс дела: в ней анализируются *элементы языка* VBA, ключевые слова и операторы, используемые для написания процедур VBA.

Для начала в качестве примера рассмотрим простую процедуру. Она хранится в модуле VBA и вычисляет сумму первых 100 целых чисел. По окончании вычислений процедура отображает сообщение с результатом.

```
Sub VBA_Demo()
' Это пример простой процедуры VBA
     Dim Total As Integer, i As Integer
     Total = 0
     For i = 1 To 100
          Total = Total + i
     Next i
     MsgBox Total
End Sub
```

В данной процедуре используются некоторые популярные элементы языка, в том числе комментарий (строка после апострофа), переменная (`Total`), два оператора присвоения (`Total = 0` и `Total = Total + i`), циклическая структура (`For-Next`) и оператор VBA (`MsgBox`). Все указанные элементы рассматриваются в следующих разделах этой главы.

Процедуры VBA не всегда управляют объектами. Например, рассмотренная выше процедура не имеет ничего общего с объектами — она оперирует цифрами.

Комментарии

Комментарий — это описательный текст, который включается в код. VBA полностью игнорирует текст комментария. Комментарии можно использовать самым произвольным образом для описаний действий в коде (предназначение оператора не всегда очевидно).

Вы можете использовать для комментария новую строку либо вставить комментарий после инструкции в той же строке. Комментарий обозначается апострофом. VBA игнорирует любой текст, введенный после апострофа — за исключением случаев, когда апостроф заключен в кавычки. Например, в следующем операторе комментарий отсутствует, хотя в нем и содержится апостроф.

```
Msg = "Can't continue"
```

Давайте рассмотрим процедуру VBA с тремя комментариями.

```
Sub Comments()
'    Эта процедура не выполняет ничего значимого
     x = 0   'x не содержит ничего
'    Отображение результата
     MsgBox x
End Sub
```

Как правило, в качестве индикатора комментария используется апостроф, однако вы также можете применить для обозначения строки комментария ключевое слово `Rem`.

```
Rem - Следующий оператор запрашивает имя файла
```

Ключевое слово `Rem` — это, по существу, пережиток старых версий BASIC; его включили в VBA из соображений совместимости. В отличие от апострофа, `Rem` используется только в начале строки, его не допускается применять в той же строке, что и инструкция.

Использование комментариев — удачная идея, но не все комментарии в равной степени предпочтительны. Чтобы комментарий был полезен, он должен содержать информацию, которая неочевидна при рассмотрении самого кода. В противном случае вы просто увеличите общий объем файла приложения. Например, следующая процедура содержит несколько комментариев, которые на самом деле никакой ценности не представляют.

```
Sub BadComments()
'    Объявление переменных
     Dim x As Integer
     Dim y As Integer
     Dim z As Integer
'    Начало процедуры
     x = 100 ' Присвоение x значения 100
     y = 200 ' Присвоение y значения 200
'    Сложение x и y и сохранение результата в z
     z = x + y
'    Отображение результата
     MsgBox z
End Sub
```

Введение кода VBA

Код VBA, который содержится в модуле VBA, состоит из инструкций. Общепринято вводить по одной инструкции в каждой строке. Однако этот стандарт не является обязательным требованием, и вы можете применить двоеточие для разделения нескольких инструкций в одной строке. В следующем примере четыре инструкции приведены в одной строке.

```
Sub OneLine()
        x=1: y=2: z=3: MsgBox x + y + z
End Sub
```

Многие программисты придерживаются мысли о том, что код легче воспринимается, если инструкции записаны по одной в каждой строке.

```
Sub OneLine()
      x=1
      y=2
      z=3
MsgBox x + y + z
End Sub
```

Строка может иметь любую длину; модуль VBA продолжается на следующей строке, когда текущая строка доходит до правой границы окна. В длинных строках допускается использование оператора продолжения строки VBA: пробел с подчеркиванием (_).

```
Sub LongLine()
      SummedValue = _
             Worksheets("Лист1").Range("A1").Value + _
             Worksheets("Лист2").Range("A1").Value
End Sub
```

При записи макросов Excel довольно часто символы подчеркивания применяют для разбиения длинных операторов на несколько строк.

После ввода инструкции редактор Visual Basic выполняет следующие действия в целях улучшения читабельности кода.

- Вставляет пробелы между операторами. К примеру, если вы введете `Ans=1+2` (без пробелов), то VBE преобразует это выражение следующим образом.

  ```
  Ans = 1 + 2
  ```

- VBA изменяет регистр символов ключевых слов, свойств и методов. Если вы введете выражение

  ```
  Result=activesheet.range("a1").value=12
  ```

  то VBA преобразует его в следующий оператор

  ```
  Result = ActiveSheet.Range("a1").Value = 12
  ```

  Обратите внимание, что текст внутри кавычек (в данном случае, `"a1"`) не изменяется.

- Так как названия переменных VBA не чувствительны к регистру, то интерпретатор по умолчанию изменяет названия всех переменных, состоящих из букв одного регистра, таким образом, чтобы их регистр соответствовал последнему введенному варианту. Например, если вы сначала определите переменную как `myvalue` (все буквы в нижнем регистре) и затем введете переменную `MyValue` (смешанный регистр), то VBA поменяет название переменной во всех остальных случаях на `MyValue`. Исключение может быть

лишь тогда, когда вы объявите переменную с помощью ключевого слова `Dim` или другого специального оператора — название переменной останется неизменным, в виде, объявленном в начале процедуры.

- VBE просматривает инструкции на наличие синтаксических ошибок. Если VBE находит ошибку, то цвет строки изменяется, а на экране может быть отображено сообщение, описывающее проблему. Используйте команду Tools⇨Options строки меню VBE, чтобы отобразить диалоговое окно Options. В этом окне задается цвет, которым выделяются ошибки (на вкладке Editor Format) и указывается необходимость отображения сообщения об ошибке (параметр Auto Syntax Check на вкладке Editor).

Ниже приведено несколько общих советов по эффективному использованию комментариев. Итак, обращайтесь к комментариям в следующих случаях:

- для краткого описания назначения каждой созданной процедуры;
- для описания изменений, которые вы вносите в процедуру.
- для указания функции или конструкции, использующихся необычным или нестандартным способом;
- для описания назначения переменных (чтобы и вы, и другие пользователи могли истолковать названия переменных, которые без комментария понять сложно);
- для описания способов, которые помогут избежать влияния внутренних ошибок Excel на ваше приложение.

Помните, что следует вставлять комментарии по мере написания кода, но не позже.

СОВЕТ

Возможно, вы захотите проверить процедуру, не включая в нее определенную инструкцию или группу инструкций. Вместо того, чтобы удалить их из кода, превратите соответствующий фрагмент в комментарий, добавив апостроф в начале строки. Тогда VBA проигнорирует инструкцию (инструкции) при выполнении процедуры. Чтобы снова превратить комментарий в инструкцию, удалите апостроф.

Панель инструментов Edit в VBE содержит несколько полезных кнопок. Выделите группу инструкций, а затем используйте кнопку Comment Block, чтобы преобразовать инструкции в комментарии. Кнопка Uncomment Block преобразовывает группу комментариев обратно в инструкции. Указанные кнопки применяются очень часто, поэтому с целью повышения удобства работы вы можете скопировать их на стандартную панель инструментов.

Переменные, типы данных и константы

Главное предназначение VBA — обработка данных. Некоторые данные сохраняются в объектах, например, диапазонах рабочих листов. Другие данные хранятся в созданных вами переменных.

Переменная представляет собой именованное место хранения данных в памяти компьютера. Переменные могут содержать данные разных *типов* — от простых логических, или булевых, значений (`True` или `False`) до больших значений с двойной точностью (см. следующий раздел). Значение присваивается переменной с помощью оператора равенства (подробнее об этом — далее в главе).

Вы существенно сможете облегчить себе жизнь, если научитесь давать переменным простые описательные имена. Однако помните, что VBA поддерживает несколько правил, ограничивающих вас в именовании переменных.

- Вы можете использовать в названиях символы букв, числа и некоторые знаки препинания, но первой в имени переменной всегда должна вводиться буква.

- VBA не различает регистры. Чтобы сделать имена переменных удобочитаемыми, программисты часто используют смешанный регистр (например, `InterestRate`, а не `interestrate`).
- Нельзя использовать в именах пробелы или точки. Чтобы сделать имена переменных более удобными для чтения, программисты вводят символ подчеркивания (`Interest_Rate`).
- Специальные символы объявления типов (`#`, `$`, `%`, `&` или `!`) также не применяются в имени переменной.
- Названия переменных ограничены длиной 254 символов — вряд ли вы в здравом уме придумаете настолько длинное название!

В приведенном далее списке содержатся отдельные примеры выражений присвоения, в которых используются различные типы переменных. Названия переменных указываются слева от знака равенства. Каждый оператор присваивает переменной слева от знака равенства значение, которое располагается справа от знака равенства.

```
x =1
InterestRate = 0.075
LoanPayoffAmount = 234089
DataEntered = False
x = x + 1
MyNum = YourNum * 1.25
UserName = "Bob Johnson"
DateStarted = #3/14/98#
```

В VBA используется очень много *зарезервированных слов*, т.е. таких слов, которые не допускается применять в качестве названий переменных или процедур. Если вы попытаетесь ввести одно из таких слов, то будет отображено сообщение об ошибке. Например, несмотря на то, что зарезервированное слово `Next` могло бы стать описательным названием многих переменных, следующая инструкция генерирует синтаксическую ошибку.

```
Next = 132
```

К сожалению, сообщения о синтаксических ошибках не всегда достаточно описательны. Указанная выше инструкция генерирует сообщение об ошибке: `Compile error: Expected Variable` (Ошибка компиляции: ожидается переменная). Ситуация не совсем понятна, поэтому при отображении такого сообщения об ошибке обращайтесь к справочной системе, чтобы убедиться в том, что имя переменной не задействовано в VBA в других целях.

Определение типов данных

VBA облегчает жизнь программистам, автоматически обрабатывая любые типы данных. Не во всех языках программирования управление данными выполняется на столь простом уровне. Например, многие языки программирования являются строго типизированными, т.е. программист должен явно объявлять тип данных каждой используемой переменной.

Тип данных указывает, в каком виде данные хранятся в памяти: как целые значения, действительные числа, текст и т.п. VBA может автоматически типизировать данные, однако это чревато негативными последствиями — медленное выполнение операций и менее эффективное использование памяти. В результате, позволяя VBA самостоятельно определять типы данных, вы можете столкнуться с проблемами выполнения больших или сложных приложений. Если вам приходится экономить каждый байт памяти, то вы должны хорошо разбираться в типах данных. Еще пре-

имуществом явного объявления типа данных всех используемых переменных является то, что хотя VBA дополнительно ищет ошибки на этапе компиляции, подобные ошибки довольно сложно выявить другими способами проверки.

В табл. 8.1 перечислены поддерживаемые в VBA типы данных (обратите внимание, всегда можно определить специальные типы данных, которые будут описаны немного позже в этой главе).

Таблица 8.1. Встроенные типы данных VBA

Тип данных	Резервируется байт	Диапазон значений
`Byte`	1 байт	От 0 до 255
`Boolean`	2 байта	`True` (Истина) или `False` (Ложь)
`Integer`	2 байта	От –32768 до 32767
`Long`	4 байта	От –2147483648 до 2147483647
`Single`	4 байта	От –3,402823E38 до –1,401298E-45 (для отрицательных значений); от 1,401298E-45 до 3,402823E38 (для положительных значений)
`Double`	8 байт	От –1,79769313486232E308 до –4,94065645841247E-324 (отрицательные числа); от 4,94065645841247E-324 до 1,79769313486232E308 (положительные числа)
`Currency`	8 байт	От –922337203685477,5808 до 922337203685477,5807
`Decimal`	14 байт	+/–79228162514264337593543950335 без десятичных знаков; +/–7,9228162514264337593543950335 с 28-ю знаками после запятой
`Date`	8 байт	С 1 января 100 года до 31 декабря 9999 года
`Object`	4 байта	Любая ссылка на объект
`String` (переменной длины)	10 байт + длина строки	От 0 до приблизительно 2 млрд.
`String` (фиксированной длины)	Длина строки	От 1 до приблизительно 65400
`Variant` (числа)	16 байт	Любое числовое значение в рамках диапазона типа данных `Double`
`Variant` (символы)	22 байта + длина строки	От 0 до приблизительно 2 млрд.
Пользовательский	Зависит от типа	Зависит от элемента

Тип данных `Decimal` впервые появился в Excel 2000 и не может использоваться в более старых версиях программы. Он достаточно необычен, поскольку его нельзя объявлять. В действительности он является “подтипом” типа `Variant`. Для преобразования типа `Variant` в десятичный тип данных (`Decimal`) необходимо использовать функцию VBA `CDec`.

Рекомендуется выбирать тот тип данных, в котором используется минимальное количество байт для хранения значений, но он также должен быть достаточным для представления максимальных значений переменных. При работе с данными в VBA скорость выполнения операций зависит от объема данных, которые обрабатываются с помощью VBA. Другими словами, чем меньше байт зарезервировано под данные, тем быстрее VBA получает доступ к данным и обрабатывает их.

Для проведения математических вычислений в рабочих листах Excel использует тип данных Double. Его советуем применять и в процессе обработки чисел в VBA для обеспечения той же точности вычислений. Для обработки целочисленных значений идеально подходит тип Integer, если, конечно, вы уверены, что используемые значения не превышают 32767. В противном случае обратитесь к типу данных Long. При управлении номерами строк в рабочем листе Excel лучше применять тип данных Long, так как количество строк в рабочем листе превышает максимальное значение, допустимое в типе данных Integer.

Объявление переменных

Если вы не объявили тип данных для переменной, используемой в процедуре VBA, по умолчанию будет задан тип данных Variant. Данные с типом Variant имеют только одно заслуживающее внимания свойство: они изменяют свой тип, в зависимости от того, какие операции над ними выполняются. В следующей процедуре показано, каким образом переменная принимает разные типы данных.

```
Sub VariantDemo()
     MyVar = "123"
     MyVar = MyVar / 2
     MyVar = "Ответ: " & MyVar
     MsgBox MyVar
End Sub
```

В процедуре VariantDemo переменная MyVar вначале выступает строкой из трех символов. Затем эта “строка” делится на два и приобретает числовой тип данных. После этого MyVar присоединяется к строке, что вызывает обратное преобразование MyVar в строку. Оператор MsgBox отображает окончательное строковое значение: *Ответ: 61,5*.

Чтобы проиллюстрировать проблемы, которые могут возникнуть при обработке типа данных Variant, рассмотрим следующую процедуру.

```
Sub VariantDemo2()
     MyVar = "123"
     MyVar = MyVar + MyVar
     MyVar = "Ответ: " & MyVar
     MsgBox MyVar
End Sub
```

При выполнении этой процедуры в окне сообщений появится сообщение *Ответ: 123123*. Наверное, это *не* тот результат, который вы ожидали. В процессе управления текстовыми данными, представленными типом Variant, оператор + выполняет конкатенацию строк.

ОПРЕДЕЛЕНИЕ ТИПА ДАННЫХ

Для определения типа данных переменной используется функция VBA TypeName. Ниже представлена модифицированная версия предыдущей процедуры. Эта процедура на каждом шаге отображает тип данных переменной MyVar. Вы увидите, что сначала переменная является строкой, затем преобразовывается в числовой тип Double и в завершение снова становится строкой.

```
Sub VariantDemo2()
     MyVar = "123"
     MsgBox TypeName(MyVar)
     MyVar = MyVar / 2
     MsgBox TypeName(MyVar)
     MyVar = "Ответ: " & MyVar
     MsgBox TypeName(MyVar)
     MsgBox MyVar
End Sub
```

Благодаря VBA преобразование типов для необъявленных переменных выполняется автоматически. Этот процесс может показаться удачным выходом из создавшейся ситуации, однако помните, что при этом уменьшается скорость обработки данных и быстрее заполняется свободная память.

Тестирование типов данных

Чтобы оценить важность определения типа данных, рассмотрим следующую процедуру, в которой выполняются циклические вычисления, а затем отображается общее время выполнения процедуры.

```
Sub TimeTest()
      Dim x As Integer, y As Integer
      Dim A As Integer, B As Integer, C As Integer
      Dim i As Integer, j As Integer
      Dim StartTime As Date, EndTime As Date
'     Сохранение времени начала вычислений
      StartTime = Timer
'     Выполнение вычислений
      x = 0
      y = 0
      For i = 1 To 5000
            For j = 1 To 1000
                  A = x + y + i
                  B = y - x - i
                  C = x - y - i
            Next j
      Next i
'     Получение времени окончания вычислений
      EndTime = Timer
'     Отображение общего времени в секундах
      MsgBox Format(EndTime - StartTime, "0.0")
End Sub
```

Эта процедура выполняется около 4,0 секунд (полученное время, в основном, зависит от быстродействия процессора). Если мы закомментируем операторы `Dim`, объявляющие типы данных (т.е. превратим операторы `Dim` в комментарии, добавив апострофы в начале соответствующих строк), то в VBA будет использоваться тип данных по умолчанию — `Variant`. Вновь запустим процедуру. Время выполнения составит 8,3 секунды — более чем в два раза превышающее исходное.

Таким образом, если вы хотите, чтобы ваши VBA-приложения работали как можно быстрее, объявляйте переменные вручную!

Объявлять каждую переменную в процедуре перед ее использованием — замечательная привычка. В процессе объявления переменной вы явно указываете ее название и тип данных. Объявление переменных предоставляет два основных преимущества.

- *Программы работают быстрее и используют память более эффективно.* Тип данных по умолчанию, `Variant`, резервирует больше памяти, чем это необходимо, и вызывает многократную проверку данных, занимающую процессорное время. Если программа точно знает тип данных, она не выполняет дополнительную проверку данных и резервирует ровно столько памяти, сколько необходимо для хранения конечных данных.

- *Объявление переменных позволяет избежать ошибок, связанных с неправильным введением имен переменных.* Предполагается, что вы используете оператор `Option Explicit`, делающий обязательным объявление всех переменных (см. следующий раздел). Предположим, что вы также в коде применяете необъявленную переменную с названием `CurrentRate`. Наряду с этим на определенном этапе процедуры вы обращаетесь к оператору `CurentRate = 0.75`. Такая ошибка в названии переменной, которую тяжело выявить, скорее всего, будет причиной неправильных результатов.

ОБЯЗАТЕЛЬНОЕ ОБЪЯВЛЕНИЕ ВСЕХ ПЕРЕМЕННЫХ

Чтобы обеспечить обязательное объявление всех используемых переменных, необходимо включить следующую строку в качестве первой инструкции в модуле VBA.

```
Option Explicit
```

Этот оператор отвечает за то, что программа будет приостанавливаться всякий раз при нахождении в ней необъявленной переменной. VBA выведет сообщение об ошибке; для продолжения выполнения процедуры вы должны будете объявить переменную.

> **СОВЕТ** Чтобы оператор `Option Explicit` автоматически был вставлен при создании нового модуля VBA, активизируйте опцию Require Variable Declaration (Обязательное объявление переменных) на вкладке Editor диалогового окна Options редактора VBE. Настоятельно рекомендуем вам не игнорировать эту установку.

Область действия переменных

Область действия переменной определяет, в каких модулях и процедурах она может использоваться. Существуют следующие типы областей действия переменных.

Область действия	Способ объявления переменной
Отдельная процедура	В процедуру включается оператор `Dim` или `Static`
Отдельный модуль	Перед первой процедурой в модуле вводится оператор `Dim` или `Private`
Все модули	Перед первой процедурой в модуле вводится оператор `Public`

Все типы области действия рассмотрены в следующих разделах.

О примерах, приводимых в этой главе

Настоящая глава содержит ряд примеров кода VBA, обычно представленных в виде простых процедур. Все примеры призваны объяснить наиболее простым образом разные концепции программирования. Во многих примерах поставленные задачи выполняются неэффективно, зачастую решить их можно иным образом. Другими словами, не стоит с точностью использовать эти примеры в собственных проектах. В следующих главах книги приводимые примеры действительно представляют *практическую* ценность.

ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ

Локальная переменная — это переменная, объявленная в процедуре. Локальные переменные могут использоваться только в процедуре, в которой они объявлены. После выполнения процедуры переменная становится невостребованной, поэтому Excel освобождает соответствующую область памяти.

Если требуется сохранить значение переменной, объявите ее как `Static` (см. далее подраздел “Переменные `Static`”).

Наиболее популярный способ объявить локальную переменную — вставить оператор `Dim` между операторами `Sub` и `End Sub`. Операторы `Dim` обычно вводятся непосредственно после оператора `Sub`, перед кодом процедуры.

Если вас интересует происхождение ключевого слова `Dim`, то несложно заметить, что это сокращение от *Dimension* (*Размерность*). В старых версиях BASIC этот оператор использовался исключительно для объявления размерности массива. В VBA ключевое слово `Dim` применяется для объявления любой переменной, а не только массивов.

В представленной далее процедуре используется шесть локальных переменных, объявленных с помощью операторов `Dim`.

```
Sub MySub()
     Dim x As Integer
     Dim First As Long
     Dim InterestRate As Single
     Dim TodaysDate As Single
     Dim UserName As String * 20
     Dim MyValue
'    -[Здесь указывается код процедуры] -
End Sub
```

Обратите внимание, что последний оператор `Dim` в этом примере объявляет не тип данных, а только саму переменную. В результате переменная приобретает тип `Variant`.

Кроме того, вы можете объявить несколько переменных, воспользовавшись одним оператором `Dim`.

```
Dim x As Integer, y As Integer, z As Integer
Dim First As Long, Last As Double
```

Другой способ указания типов данных для переменных

Как и в большинстве других версий BASIC, язык VBA позволяет присоединить символ к названию переменной, чтобы указать ее тип данных. Например, вы можете объявить переменную `MyVar` как целое число, добавив к ее названию символ `%`.

```
Dim MyVar%
```

Символы объявления типов данных представлены для большинства типов данных VBA (отсутствующие в таблице типы данных не имеют собственного символа объявления типа).

Тип данных	Символ объявления типа
`Integer`	`%`
`Long`	`&`
`Single`	`!`
`Double`	`#`
`Currency`	`@`
`String`	`$`

Этот метод типизации данных — пережиток старых версий BASIC; эффективнее объявлять переменные с помощью остальных методов, описанных в настоящей главе.

В отличие от других языков программирования, в VBA нельзя объявить тип данных одновременно для группы переменных, разделив переменные запятыми. Например, следующий оператор, является корректным, однако он не объявляет все переменные как `Integer`.

```
Dim i, j, k As Integer
```

В VBA только `k` объявляется как целое, другие же переменные получают тип `Variant`. Чтобы объявить `i`, `j` и `k` как `Integer`, используйте такой оператор.

```
Dim i As Integer, j As Integer, k As Integer
```

Если переменная объявлена как локальная, другие процедуры в том же модуле могут использовать подобное имя, но каждый экземпляр переменной считается уникальным в своей процедуре.

Как правило, локальные переменные — самые эффективные, так как VBA освобождает память, которую они используют, после окончания выполнения процедуры.

ПЕРЕМЕННЫЕ УРОВНЯ МОДУЛЯ

Иногда необходимо, чтобы переменная была доступна во всех процедурах модуля. В таком случае объявите переменную *перед* первой процедурой модуля (за пределами процедур или функций).

В приведенном ниже примере оператор `Dim` — первая инструкция в модуле. Обе процедуры `MySub` и `YourSub` имеют доступ к переменной `CurrentValue`.

```
Dim CurrentValue As Integer

Sub MySub()
'    -[Здесь вводится текст процедуры] -
End Sub

Sub YourSub()
'    -[Здесь вводится текст процедуры] -
End Sub
```

Значение переменной уровня модуля не изменяется к окончанию выполнения процедуры.

ПЕРЕМЕННЫЕ PUBLIC

Чтобы сделать переменную доступной во всех процедурах всех модулей VBA проекта, необходимо объявить переменную на уровне модуля с помощью ключевого слова `Public`, а не `Dim`.

```
Public CurrentRate as Long
```

Ключевое слово `Public` делает переменную `CurrentRate` доступной для любой процедуры проекта, даже для процедур, которые располагаются в других модулях проекта. Этот оператор следует вставить перед первой процедурой модуля. Более того, подобный код объявления переменных должен вводиться в стандартном модуле VBA, а не в коде модуля листа или формы.

ПЕРЕМЕННЫЕ STATIC

Переменные `Static` — особый случай. Они объявляются на уровне процедуры и сохраняют свое значение после окончания процедуры.

```
Sub MySub()
     Static Counter As Integer
     -[Здесь водится текст процедуры] -
End Sub
```

Работа с константами

Значение переменной может изменяться при выполнении процедуры (часто так и случается, поэтому она и называется "переменной"). Иногда же необходимо использовать именованное значение или строку, которая никогда не меняется — константу.

ОБЪЯВЛЕНИЕ КОНСТАНТ

Константы объявляются с помощью оператора `Const`. Ниже приведено несколько примеров.

```
Const NumQuarters as Integer = 4
Const Rate = .0725, Period = 12
Const ModName as String = "Budget Macros"
Public Const AppName as String = "Budget Application"
```

Во втором примере тип данных не объявлен. Следовательно, указанные две константы имеют тип `Variant`. Так как константы никогда не изменяют своего значения, обычно их объявляют в виде конкретного типа данных.

Как и переменные, константы также имеют область действия. Если требуется, чтобы константа была доступна только в одной процедуре, объявите ее после оператора `Sub` или `Function` — она станет локальной. Вы сделаете константу доступной для всех процедур в модуле, если объявите ее перед первой процедурой модуля. Чтобы сделать константу доступной для всех модулей рабочей книги, используйте ключевое слово `Public` и объявите константу перед первой процедурой модуля.

```
Public Const InterestRate As Double = 0.0725
```



При попытке изменить значение константы в процедуре VBA вы получите ошибку (как того и следовало ожидать). Константа — это постоянное значение, а не переменная.

Использование констант в программе вместо строго запрограммированных значений или строк — удачный ход программирования. Например, если процедура несколько раз ссылается на определенное значение, например, процентную ставку, следует объявить это значение как константу и использовать в выражениях имя константы, а не ее значение. Такой прием не только делает программу более удобной для восприятия, но и облегчает последующее изменение кода — достаточно поменять только одну инструкцию, а не несколько.

ИСПОЛЬЗОВАНИЕ ПРЕДОПРЕДЕЛЕННЫХ КОНСТАНТ

В Excel и VBA существует целый ряд предопределенных констант, которые можно использовать без объявления. Вам даже не обязательно знать значение этих констант для их применения. При записи макросов обычно используются константы, а не значения. В следующей процедуре для изменения ориентации страницы активного листа на альбомную применена встроенная константа (`xlLandscape`).

```
Sub SetToLandscape()
     ActiveSheet.PageSetup.Orientation = xlLandscape
End Sub
```

Правила именования переменных

Некоторые программисты называют переменные так, что типы данных легко определить по одному только названию. Однако использование такого приема усложняет восприятие кода.

Распространенное правило именования связано с использованием стандартной приставки в нижнем регистре в названии переменной. Например, булеву переменную, которая контролирует сохранение рабочей книги, можно назвать `bWasSaved`. Таким образом, понятно, что это переменная типа `Boolean`. В приведенной ниже таблице перечислены стандартные префиксы для некоторых типов данных.

Тип данных	Префикс
`Boolean`	`b`
`Integer`	`i`
`Long`	`l`
`Single`	`s`
`Double`	`d`
`Currency`	`c`
`Date/Time`	`dt`
`String`	`str`
`Object`	`obj`
`Variant`	`v`
Пользовательский	`u`

Константа `xlLandscape` была обнаружена путем записи макроса. Описание константы также приводится в справочной системе. Кроме того, если у вас включен параметр AutoList Members, то вы можете получить помощь непосредственно при вводе кода. Во многих случаях VBA автоматически перечисляет все константы, присваиваемые определенному свойству.

Фактическое значение переменной `xlLandscape` равно 2. Еще одна встроенная константа для изменения ориентации страницы — `xlPortrait` — имеет значение 1. Очевидно, что при использовании встроенных констант не обязательно знать их значения.

НА ЗАМЕТКУ

Окно Object Browser, рассмотренное в главе 7, содержит список всех констант Excel и VBA. Чтобы запустить утилиту Object Browser, в VBE нажмите <F2>.

Управление строками

Как и Excel, VBA может работать не только с числами, но и с текстом (строками). В VBA представлено два типа строк.

- *Строки фиксированной длины* объявляются с определенным количеством символов. Максимальная длина строки составляет 65535 символов.
- *Строки переменной длины* теоретически могут вмещать до 2 млрд. символов.

Каждый символ в строке занимает 1 байт памяти, к тому же, память дополнительно используется для хранения заголовка строки. При объявлении строки с помощью оператора `Dim` вы можете определить ее длину, если она вам известна (задать тип строки с фиксированной длиной), или активизировать динамическую обработку дли-

ны строки (задать тип строки переменной длины). Работа со строками фиксированной длины несколько эффективнее с точки зрения использования памяти.

В следующем примере переменная `MyString` объявляется как строка с максимальной длиной 50 символов. `YourString` тоже объявлена как строка, но она имеет переменную длину.

```
Dim MyString As String * 50
Dim YourString As String
```

Работа с датами

Конечно, для хранения даты можно использовать строку, но над ней нельзя выполнять вычисления. Применение специального типа данных `Date` — лучший способ управления датами.

Переменная, определенная как `Date`, занимает 8 байт памяти и может содержать даты в диапазоне с 1 января 100 года до 31 декабря 9999 года. Это диапазон почти в 10000 лет — более чем достаточно даже для самого невероятного финансового прогноза! Тип данных `Date` также применяется для хранения значений времени. В VBA дата и время определяются как значения, заключенные между знаками # (см. далее).

Ошибка дат в Excel

В Excel (это общеизвестный недостаток) используется неправильное предположение, что 1900 год — високосный. Поэтому в Excel можно представить дату 29 февраля 1900 года. Excel принимает следующую информацию и отображает результат как 29-й день февраля 1900-го года.

```
=ДАТА(1900;2;29)
```

В VBA не существует подобной ошибки. Эквивалентом функции `ДАТА` Excel в VBA является `DateSerial`. Представленное далее выражение (корректно) возвращает 1 марта 1900 года.

```
=DateSerial(1900,2,29)
```

Следовательно, система представления дат в Excel не соответствует системе представления дат в VBA. Эти две системы возвращают различные значения для дат в диапазоне с 1 января 1900 года до 1 марта 1900 года.

НА ЗАМЕТКУ: Диапазон дат, которые можно обрабатывать в VBA, намного шире, чем собственный диапазон дат Excel, который начинается с 1 января 1900 года. Поэтому следите, чтобы в рабочем листе не использовались данные, которые находятся за пределами приемлемого диапазона дат Excel.

Ниже приведены отдельные примеры объявления переменных и констант с типом данных `Date`.

```
Dim Today As Date
Dim StartTime As Date
Const FirstDay As Date = #1/1/2001#
Const Noon = #12:00:00#
```

ВНИМАНИЕ: Константы дат всегда определяются с помощью формата месяц/день/год, даже если система установлена на отображение данных в другом формате (например, день/месяц/год).

Если вы используете для отображения даты окно сообщений, дата будет представлена в соответствии с коротким форматом даты, установленным в системе. Аналогично, время отображается согласно системному формату (12-часовому или 24-часовому). Вы можете модифицировать системные настройки с помощью аплета Язык и региональные стандарты папки Панель управления в Windows.

НА КОМПАКТ-ДИСКЕ

На прилагаемом к книге компакт-диске вы найдете пакет Extended Data Functions. Этот пакет создан с помощью VBA. Он добавляет в Excel новые функции рабочего листа, которые предоставляют возможность создавать формулы, управляющие датами до 1 января 1900 года.

Операторы присвоения

Оператор присвоения — это инструкция VBA, выполняющая математическое вычисление и присваивающая результат переменной или объекту. В справочной системе Excel *выражение* определяется как комбинация ключевых слов, операторов, переменных и констант. Эта комбинация возвращает в результате строку, число или объект. Выражение может осуществлять вычисление, обрабатывать символы или тестировать данные.

Нельзя придумать лучшего определения. Большое количество операций, выполняемых в VBA, связано с разработкой (и отладкой) выражений.

Если вы знаете, как создавать формулы в Excel, то у вас не будет возникать проблем с созданием выражений в VBA. В формуле рабочего листа Excel результат отображается в ячейке. С другой стороны, выражение VBA может присваивать значение переменной или использоваться как значение свойства.

В VBA оператором присвоения выступает знак равенства (=). Ниже приведены примеры использования операторов присвоения (выражения приводятся справа от знака равенства).

```
x = 1
x = x + 1
x = (y * 2) / (z * 2)
FileOpen = True
FileOpen = Not FileOpen
Range("The Year").Value = 2001
```

СОВЕТ

Выражения могут быть очень сложными. Чтобы сделать длинные выражения более удобными для восприятия, используйте символ продолжения строки (пробел с подчеркиванием).

Зачастую в выражениях применяются функции. Это могут быть встроенные функции VBA, функции рабочих листов Excel или специальные функции, разработанные в VBA.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Встроенные функции VBA рассматриваются далее в этой главе.

В VBA математические операторы играют главную роль. Известные вам операторы описывают математические операции, в том числе сложение (+), умножение (*), деление (/), вычитание (−), возведение в степень (^) и конкатенацию строк (&). Менее знакомые операторы: обратная косая черта (\) — используется в целочисленном делении, оператор `Mod` — применяется при определении модуля числа. Оператор `Mod` возвращает остаток от деления одного числа на другое. Например, следующее выражение возвращает 2.

```
17 Mod 3
```

VBA поддерживает операторы сравнения, которые применяются в формулах Excel: равно (=), больше (>), меньше (<), больше или равно (>=), меньше или равно (<=) и не равно (<>).

Кроме того, VBA представляет полный набор булевых операторов (табл. 8.2). Полную информацию о них (с примерами) вы найдете в справочной системе по VBA.

Таблица 8.2. Булевы операторы VBA

Оператор	Действие
`Not`	Логическое отрицание выражения
`And`	Логическая конъюнкция двух выражений
`Or`	Логическая дизъюнкция двух выражений
`XoR`	Логическое отрицание двух выражений
`Eqv`	Логическая эквивалентность двух выражений
`Imp`	Логическая импликация двух выражений

Порядок приоритетности выполнения операторов VBA такой же, как и в Excel. Конечно, вы можете добавить скобки, чтобы изменить приоритет операторов по умолчанию.

В следующей инструкции используется оператор `Not` для отображения линий сетки в активном окне. Свойство `DisplayGridLines` принимает значение `True` или `False`. Следовательно, применение оператора `Not` изменяет `True` на `False`, а `False` — на `True`.

```
ActiveWindow.DisplayGridLines = _
     Not ActiveWindow.DisplayGridLines
```

Представленное далее выражение осуществляет логическую операцию `And`. Оператор `MsgBox` отображает `True`, только если `Лист1` — активный лист *и* активная ячейка находится в строке 1. Если одно *или* оба этих условия неверны, оператор `MsgBox` отображает `False`.

```
MsgBox ActiveSheet.Name = "Лист1" And ActiveCell.Row = 1
```

Следующее выражение осуществляет логическую операцию `Or`. Оператор `MsgBox` отображает `True`, если активен `Лист1` *или* `Лист2`.

```
MsgBox ActiveSheet.Name = "Лист1" _
     Or ActiveSheet.Name = "Лист2"
```

Массивы

Массив — это группа элементов одного типа, которые имеют общее имя; на конкретный элемент массива ссылаются, используя имя массива и индекс. Например, можно определить массив из 12-ти строк так, чтобы каждая переменная соответствовала названию месяца. Если вы назовете массив `MonthNames`, то можете обратиться к первому элементу массива как `MonthNames(0)`, ко второму — как `MonthNames(1)` и т.д., до `MonthNames(11)`.

Объявление массивов

Массив, как и обычные переменные, объявляется с помощью операторов Dim или Public. Кроме того, можно определить количество элементов в массиве: введите нижний индекс, ключевое слово To и верхний индекс — вся конструкция будет заключена в скобки. Например, объявить массив, содержащий ровно 100 целых чисел можно следующим образом.

```
Dim MyArray(1 To 100) As Integer
```

СОВЕТ

При объявлении массива обязательно следует указывать только верхний индекс, тогда VBA установит нижний индекс равным нулю. Следовательно, два следующих оператора приведут к одинаковым результатам.

```
Dim MyArray(0 To 100) As Integer
Dim MyArray(100) As Integer
```

В обоих случаях массив состоит из 101-го элемента.

Если вы хотите, чтобы в качестве первого индекса всех массивов использовалась единица, то вам необходимо перед первой процедурой модуля сделать следующее объявление.

```
Option Base 1
```

Объявление многомерных массивов

В примерах массивов в предыдущем разделе использовались одномерные массивы. Массивы VBA могут иметь до 60-ти измерений, хотя на самом деле используется не более трех (трехмерные массивы). Показанный ниже оператор объявляет двухмерный 100-элементный массив целых чисел.

```
Dim MyArray(1 To 10, 1 To 10) As Integer
```

Этот массив можно рассматривать как матрицу значений 10×10. Чтобы обратиться к конкретному элементу двухмерного массива, используйте два индекса. Например, таким образом присваивается значение элементу предыдущего массива.

```
MyArray(3, 4) = 125
```

Трехмерный массив можно рассматривать как куб, но не существует способа визуально представить данные массива, в котором больше трех измерений.

Динамический массив не имеет предопределенного количества элементов. Динамический массив объявляется с незаполненными значениями в скобках.

```
Dim MyArray() As Integer
```

Тем не менее, прежде чем динамический массив можно будет использовать в программе, необходимо обратиться к оператору ReDim, указывающему VBA, сколько элементов находится в массиве (или ReDim Preserve, если вы решили сохранить текущую длину массива). Оператор ReDim можно использовать сколько угодно раз, изменяя, если требуется, размер массива.

Массивы будут рассмотрены далее в этой главе при обсуждении циклов.

Переменные объектов

Переменная объекта — это переменная, представляющая целый объект, например, диапазон или рабочий лист. Переменные объектов имеют особое значение по следующим причинам:

- значительно упрощают программу;
- ускоряют выполнение программы.

Переменные объектов, как и обычные переменные, объявляются с помощью оператора `Dim` или `Public`. Например, в следующем операторе переменная `InputArea` объявляется как объект `Range`.

```
Public InputArea As Range
```

Чтобы узнать, каким образом переменные объектов упрощают программу, проанализируем процедуру, написанную без их использования.

```
Sub NoObjVar()
     Worksheets("Лист1").Range("A1").Value = 124
     Worksheets("Лист1").Range("A1").Font.Bold = True
     Worksheets("Лист1").Range("A1").Font.Italic = True
End Sub
```

Эта процедура вводит значение в ячейку `A1` листа `Лист1` активной рабочей книги, а затем делает начертание содержимого ячейки полужирным и курсивным. В примере введено много кода для решения такой простой задачи. Чтобы упростить ее, сведите процедуру к использованию объектной переменной.

```
Sub ObjVar()
     Dim MyCell As Range
     Set MyCell = Worksheets("Лист1").Range("A1")
     MyCell.Value = 124
     MyCell.Font.Bold = True
     MyCell.Font.Italic = True
End Sub
```

После объявления переменной `MyCell` как объекта `Range` оператор `Set` присваивает ей сам объект. В результате в следующих операторах используется упрощенная ссылка `MyCell` вместо длинной `Worksheets("Лист1").Range("A1")`.

СОВЕТ

После присвоения переменной объекта VBA получает доступ к нему быстрее, чем с помощью непосредственной ссылки на объект. Поэтому, если вам важна скорость выполнения операции, используйте переменные объектов. Удобно рассматривать этот феномен в категориях узлов. Каждый раз, когда VBA встречает новый узел (например, `Sheets(1).Range("A1")`), ему требуется определенное время на расшифровку ссылки. Применение переменной объекта уменьшает количество данных для обработки. Чем меньше узлов указано в ссылке, тем быстрее обрабатывается последняя. Другой способ увеличить скорость выполнения программы — использовать конструкцию `With-End With`, что также уменьшает количество обрабатываемых узлов. Эта конструкция рассмотрена далее в главе.

Настоящая ценность переменных объектов станет вам понятной после рассмотрения циклов.

Пользовательские типы данных

VBA позволяет создавать специальные, или *пользовательские*, типы данных (эта концепция напоминает записи Pascal или структуры C). Определенный пользователем тип данных может облегчить управление некоторыми типами данных. Например, если приложение обрабатывает сведения о клиенте, то можно создать пользовательский тип данных с названием `CustomerInfo`.

```
Type CustomerInfo
     Company As String * 25
     Contact As String * 15
     RegionCode As Integer
     Sales As Long
End Type
```

Пользовательские типы данных определяются вверху модуля перед началом введения кода процедур.

Если пользовательский тип данных уже создан, для объявления переменной этого типа примените оператор `Dim`. Обычно пользовательский тип данных определяется для массивов.

```
Dim Customers(1 To 100) As CustomerInfo
```

Все 100 элементов этого массива состоят из четырех компонентов (как указано в пользовательском типе данных — `CustomerInfo`). Вы можете сослаться на конкретный компонент элемента следующим образом.

```
Customers(1).Company = "Acme Tools"
Customers(1).Contact = "Tim Robertson"
Customers(1).RegionCode = 3
Customers(1).Sales = 150677
```

Вам также предоставлена возможность управлять элементом массива как одним целым. Например, чтобы скопировать информацию из `Customers(1)` в `Customers(2)`, используется следующая инструкция.

```
Customers(2) = Customers(1)
```

Предыдущий пример эквивалентен приведенному ниже блоку инструкций.

```
Customers(2).Company = Customers(1).Company
Customers(2).Contact = Customers(1).Contact
Customers(2).RegionCode = Customers(1).RegionCode
Customers(2).Sales = Customers(1).Sales
```

Встроенные функции

Как и в большинстве других языков программирования, в VBA есть ряд встроенных функций, упрощающих вычисления и операции. Часто функции позволяют выполнить операции, которые по-другому осуществить сложно или вообще невозможно. Многие функции VBA подобны (или идентичны) функциям Excel. Например, функция VBA `UCase`, преобразующая строку в верхний регистр, эквивалентна функции Excel `ПРОПИСН`.

Приложение Б содержит полный список функций VBA с их кратким описанием. Все функции подробно рассмотрены в справочной системе.

СОВЕТ

Чтобы получить список функций VBA при написании кода, введите **VBA** и точку (**.**). VBE отображает список всех вложенных в объект VBA объектов, включая функции (рис. 8.1). Функции обозначаются зеленым значком. Если этот способ вам недоступен, проверьте, включен ли параметр Auto List Members. Для этого выполните команду Tools⇨Options, а затем щелкните на вкладке Editor.

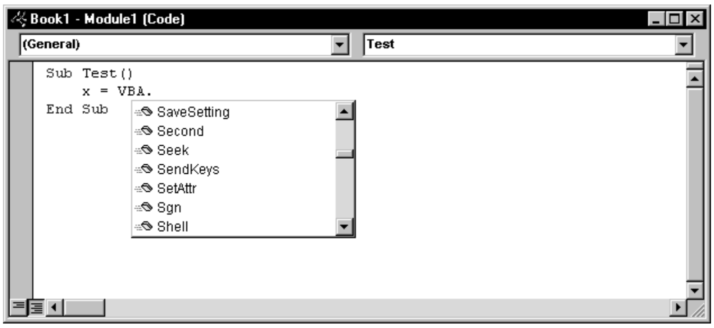


Рис. 8.1. Отображение списка функций VBA в VBE

Функции VBA используются в выражениях почти так же, как и функции Excel в формулах рабочего листа. Например, можно создавать вложенные функции VBA.

Ниже приведена простая процедура, которая вычисляет квадратный корень переменной с помощью функции `Sqr` VBA, сохраняет результат в другой переменной и затем отображает результат.

```
Sub ShowRoot()
    MyValue = 25
    SquareRoot = Sqr(MyValue)
    MsgBox SquareRoot
End Sub
```

Функция `Sqr` VBA эквивалентна функции рабочего листа `КОРЕНЬ` в Excel.

Вы можете использовать ряд (но не все) функций Excel в коде VBA. Объект `WorksheetFunction`, который содержится в объекте `Application`, располагает всеми функциями рабочего листа, которые можно вызвать в процедуре VBA.

Чтобы использовать функцию Excel в операторе VBA, перед названием функции введите следующее выражение.

```
Application.WorksheetFunction
```

Приведенный ниже пример показывает, как используется функция Excel в процедуре VBA. Нечасто используемая функция Excel `ROMAN` преобразует десятичное число в римское.

```
Sub ShowRoman()
    DecValue = 2001
    RomanValue = Application.WorksheetFunction.Roman(DecValue)
    MsgBox RomanValue
End Sub
```

Функция MsgBox

Функция MsgBox — одна из самых полезных в VBA. Во многих примерах настоящей главы она используется для отображения значения переменной.

Данная функция часто является достойной заменой простой форме. Кроме того, это превосходный инструмент отладки, так как вы можете в любое время вставить функцию MsgBox, чтобы приостановить программу и отобразить результат вычисления или присвоения.

Как правило, функции возвращают одно значение, которое можно присвоить переменной. Функция MsgBox не только возвращает значение, но и отображает диалоговое окно, в котором пользователь может выполнять определенные действия. Значение, возвращаемое функцией MsgBox, является ответом пользователя на отображенный запрос. Вы можете применить функцию MsgBox, если вас не интересует ответ пользователя, однако необходимо отобразить сообщение.

Формальный синтаксис функции MsgBox требует использования пяти аргументов (аргументы в квадратных скобках — необязательные):

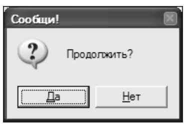
```
MsgBox(сообщение[, кнопки][, заголовок][, файл_справки, контекст])
```

- *сообщение* (обязательный аргумент) — сообщение, которое отображается в диалоговом окне.
- *кнопки* (необязательный аргумент) — значение, определяющее, какие кнопки и пиктограммы (если нужно) отображаются в окне сообщения. Применяйте встроенные константы (например, vbYesNo).
- *заголовок* (необязательный аргумент) — текст, который отображается в строке заголовка окна сообщения. По умолчанию используется текст Microsoft Excel.
- *файл_справки* (необязательный аргумент) — название файла справки, соответствующего окну сообщения.
- *контекст* (необязательный аргумент) — контекстный идентификатор раздела справки. Представляет конкретный раздел справки для отображения.

Вы можете присвоить полученное значение переменной либо использовать функцию без оператора присвоения. В приведенном ниже примере результат присваивается переменной Ans.

```
Ans = MsgBox("Продолжить?", vbYesNo + vbQuestion, "Сообщи!")
If Ans = vbNo Then Exit Sub
```

Обратите внимание, что в качестве аргумента кнопки применена сумма двух встроенных констант (vbYesNo + vbQuestion). Использование vbYesNo приводит к отображению в окне сообщения двух кнопок: Да и Нет. Добавление к аргументу значения vbQuestion вызывает вставку в окно знака вопроса (см. рисунок). Когда выполняется первый оператор, Ans содержит одно из двух значений, представленных константами vbYes или vbNo. В данном примере если пользователь щелкает на кнопке Нет, то процедура заканчивается.



Дополнительную информацию вы найдете в справочной системе, где перечислены все константы, которые можно использовать в качестве аргументов функции MsgBox.

При выполнении данной процедуры функция MsgBox отображает строку MMI. Любители старых фильмов часто приходят в отчаяние, когда узнают, что Excel не располагает функцией, преобразующей римское число в его десятичный эквивалент.

Важно понимать, что вы не можете использовать функции Excel, для которых в VBA представлены эквивалентные функции. Например, VBA не позволяет получить доступ к функции `КОРЕНЬ` (`SQRT`), так как в VBA имеется собственная версия этой функции: `Sqr`. Таким образом, следующий оператор выдает ошибку.

```
MsgBox Application.WorksheetFunction.Sqrt(123) 'ошибка
```

Как описано в главе 10, VBA можно использовать для создания специальных функций рабочих листов, работающих точно так же, как встроенные функции Excel.

Управление объектами и коллекциями

Как программист вы проведете много времени, работая с объектами и коллекциями. Поэтому вам необходимо знать самые эффективные способы написания кода для обработки объектов и коллекций. VBA предлагает две конструкции, которые помогут вам упростить управление объектами и коллекциями.

- Конструкции `With-End With`.
- Конструкции `For Each-Next`.

Конструкция With-End With

Конструкция `With-End With` позволяет выполнять несколько операций над одним объектом. Чтобы понять, как она работает, проанализируйте следующую процедуру, которая изменяет пять свойств выделенного объекта (подразумевается, что выделен объект `Range`).

```
Sub ChangeFont1()
     Selection.Font.Name = "Times New Roman"
     Selection.Font.FontStyle = "Bold Italic"
     Selection.Font.Size = 12
     Selection.Font.Underline = xlUnderlineStyleSingle
     Selection.Font.ColorIndex = 5
End Sub
```

Эту процедуру можно переписать с помощью конструкции `With-End With`. Процедура, показанная ниже, работает точно так же, как и предыдущая.

```
Sub ChangeFont2()
     With Selection.Font
          .Name = "Times New Roman"
          .FontStyle = "Bold Italic"
          .Size = 12
          .Underline = xlUnderlineStyleSingle
          .ColorIndex = 5
     End With
End Sub
```

Некоторые считают, что второй вариант этой процедуры читать сложнее. Однако помните, что целью изменений является увеличение скорости выполнения операций. Первый вариант более прямолинейный и его легче понять, но процедура, использующая для изменения нескольких свойств одного объекта конструкцию `With-End With`, помогает повысить эффективность выполнения кода по сравнению с эквивалентной ей процедурой, которая явно ссылается на объект в каждом операторе.

При записи макроса VBA в Excel конструкция `With-End With` применяется при каждой удобной возможности. Чтобы посмотреть удачный пример этой конструкции, проанализируйте последовательность действий при изменении параметров страницы с помощью команды Файл⇨Параметры страницы.

Конструкция For Each-Next

Итак, вы уже знаете, что коллекция — это группа однородных объектов. Например, коллекция `Workbooks` — это коллекция всех открытых рабочих книг `Workbook`. Существует ряд других коллекций, с которыми вы можете работать. Чтобы правильно использовать конструкцию `For Each-Next`, необязательно знать, сколько элементов насчитывает коллекция.

Предположим, вы решили выполнить действие над всеми объектами коллекции, или вам необходимо оценить все объекты коллекции и совершить действие при выполнении определенных условий. Это идеальная ситуация для применения конструкции `For Each-Next`.

Синтаксис конструкции `For Each-Next` приведен ниже.

```
For Each элемент In группа
     [инструкции]
     [Exit For]
     [инструкции]
Next [элемент]
```

Следующая процедура использует конструкцию `For Each-Next` для обращения поочередно к каждому из шести членов массива фиксированной длины.

```
Sub Macro1()
     Dim MyArray(5)
     For i = 0 To 5
          MyArray(i) = Rnd
     Next i
     For Each n In MyArray
          Debug.Print n
     Next n
End Sub
```

В процедуре, приведенной далее, применена конструкция `For Each-Next` по отношению к коллекции `Sheets` активной рабочей книги. При выполнении этой процедуры функция `MsgBox` отображает свойство `Name` каждого рабочего листа. (Если в активной рабочей книге пять листов, функция `MsgBox` вызывается пять раз.)

```
Sub CountSheets()
     Dim Item as WorkSheet
     For Each Item In ActiveWorkBook.WorkSheets
          MsgBox Item.Name
     Next Item
End Sub
```

В предыдущем примере `Item` — это переменная объекта (точнее, объект `Worksheet`). В названии `Item` не заключен особый смысл; вместо него можно использовать любое корректное название переменной.

В следующем примере конструкция `For Each-Next` используется для циклического просмотра всех объектов коллекции `Windows`.

```
Sub HiddenWindows()
     Dim AllVisible As Boolean
```

```
    Dim Item As Window
    AllVisible = True
    For Each Item In Windows
        If Item.Visible = False Then
            AllVisible = False
            Exit For
        End If
    Next Item
    MsgBox AllVisible
End Sub
```

Если окно скрыто, то значение переменной `AllVisible` изменяется на `False` и процедура выходит из цикла `For Each-Next`. В окне сообщения отображается `True` (если все окна видимы) и `False` (если хотя бы одно окно скрыто). Оператор `Exit For` необязателен. Он предоставляет способ досрочно выйти из цикла `For Each-Next`. Обычно заданная конструкция применяется вместе с оператором `If-Then` (см. далее в этой главе).

Ниже приведен пример закрытия всех рабочих книг, кроме активной. Процедура использует конструкцию `If-Then` для оценки каждой рабочей книги в коллекции `Workbooks`.

```
Sub CloseInActive()
    Dim Book as Workbook
    For Each Book In Workbooks
        If Book.Name <> ActiveWorkbook.Name Then Book.Close
    Next Book
End Sub
```

Последний представленный в этом разделе пример `For Each-Next` выполняется после того, как пользователь выделит диапазон ячеек. В данном случае объект `Selection` играет роль коллекции, состоящей из объектов `Range`, так как каждая ячейка в выделенном диапазоне представляет собой объект `Range`. Процедура оценивает каждую ячейку и использует функцию VBA `UCase` для преобразования текста, располагающегося в ней, в верхний регистр (ячейки с числовыми данными остаются неизмененными).

```
Sub MakeUpperCase()
    Dim Cell as Range
    For Each Cell In Selection
        Cell.Value = UCase(Cell.Value)
    Next Cell
End Sub
```

Контроль за выполнением кода

Некоторые процедуры VBA начинают выполняться с первых строк кода. Этот процесс производится построчно до самого конца процедуры (например, так всегда работают макросы, записанные при выполнении действий). Однако порой необходимо контролировать последовательность операций, пропуская отдельные операторы, повторно выполняя некоторые команды и проверяя условия для определения следующего действия, выполняемого процедурой.

В предыдущем разделе описана конструкция `For Each-Next`, которая является циклической структурой. В настоящем разделе речь пойдет о дополнительных способах контроля за выполнением процедур, написанных на VBA.

- Операторы `GoTo`.
- Конструкции `If-Then`.

- Конструкции `Select Case`.
- Циклы `For-Next`.
- Циклы `Do While`.
- Циклы `Do Until`.

Операторы GoTo

Самый простой способ изменить последовательность операций в коде — использовать оператор `GoTo`. Он перенаправляет ход выполнения программы на новую инструкцию, которая помечена специальным образом (текстовая строка, заканчивающаяся двоеточием, или число без двоеточия, указанные перед инструкцией). Процедуры VBA могут содержать любое количество меток, а оператор `GoTo` не определяет переход за пределы процедуры.

В приведенной ниже процедуре применена функция VBA `InputBox` для получения имени пользователя. Если имя пользователя отличается от `Ховард`, то процедура переходит к метке `WrongName`, на чем заканчивает свою работу. В противном случае процедура выполняет дополнительные операции. Оператор `Exit Sub` заканчивает выполнение процедуры.

```
Sub GoToDemo()
     UserName = InputBox("Введите свое имя:")
     If UserName <> "Ховард" Then GoTo WrongName
     MsgBox ("Привет, Ховард.")
'    -[Здесь вводится дополнительный код]
     Exit Sub
WrongName:
     MsgBox "Извините, эту процедуру может запускать только Ховард."
End Sub
```

Представленная процедура работает, но оператор `GoTo`, как правило, используется, если другого способа выполнить действие просто не существует. Единственной ситуацией, когда оператор `GoTo` в VBA *действительно* необходим, является перехват ошибок (см. главу 9).

Конструкция If-Then

Вероятно, конструкция `If-Then` чаще остальных используется для группировки инструкций VBA. Эта популярная конструкция наделяет приложения способностью принимать решения. Такая способность является ключевой при создании эффективных программ. Удачное приложение Excel, по существу, сводится к принятию правильного решения и выполнению соответствующих действий.

Стандартный синтаксис конструкции `If-Then` таков.

```
If условие Then инструкции_истина [Else инструкции_ложь]
```

Конструкция `If-Then` используется для выполнения одного или более операторов при справедливости заданного условия. Оператор `Else` необязателен. Он позволяет выполнять одну или более инструкций в случае несправедливости условия.

В описанной ниже процедуре применена структура `If-Then` без оператора `Else`. Пример связан с управлением временными данными. VBA использует систему дат и времени, похожую на задействованную в Excel. Время дня выражается дробным числом — например, полдень представлен как `0.5`. Функция VBA `Time` возвращает значение, представляющее время в формате системных часов. В следующем примере со-

общение отображается, если текущее время меньше полудня. Если текущее системное время больше или равно `0.5`, то процедура заканчивается, и ничего не происходит.

```
Sub GreetMe1()
    If Time < 0.5 Then MsgBox "Доброе утро"
End Sub
```

Вы можете отобразить другое приветствие, если текущее время больше полудня. Для этого добавьте еще один оператор `If-Then`.

```
Sub GreetMe2()
    If Time < 0.5 Then MsgBox "Доброе утро"
    If Time >= 0.5 Then MsgBox "Добрый день"
End Sub
```

Обратите внимание, что для второй конструкции `If-Then` использован оператор >= (больше или равно), — таким образом учитывается тот редкий случай, когда время на часах точно равно 12.00.

Другой подход — использовать оператор `Else` конструкции `If-Then`.

```
Sub GreetMe3()
    If Time < 0.5 Then MsgBox "Доброе утро" Else _
      MsgBox "Добрый день"
End Sub
```

В данном случае введен символ продолжения строки, а `If-Then-Else` является одним оператором.

Если вам необходимо расширить процедуру до обработки трех условий (например, утро, день и вечер), то можете использовать либо три оператора `If-Then`, либо вложенную структуру `If-Then-Else`. Первый вариант значительно проще.

```
Sub GreetMe4()
   If Time < 0.5 Then MsgBox "Доброе утро"
   If Time >= 0.5 And Time < 0.75 Then MsgBox "Добрый день"
   If Time >= 0.75 Then MsgBox "Добрый вечер"
End Sub
```

Значение `0.75` представляет время 18:00 — три четверти суток и тот момент, когда день переходит в вечер.

В предыдущих примерах каждая инструкция в процедуре выполняется, даже утром. Для эффективности следует включить структуру, заканчивающую процедуру, когда одно из условий выполняется. Если, например, отображается сообщение `Доброе утро`, то процедура заканчивается без проверки других, излишних условий. Конечно, разница в скорости выполнения обеих процедур несущественна, если вы разрабатываете такую крохотную процедуру, как рассматриваемая. Но в более сложных приложениях все же рекомендуется учесть следующий синтаксис.

```
If условие Then
    [операторы_истина]
[Elseif условие-n Then
    [альтернативные_операторы]]
[Else
    [операторы_по_умолчанию]]
End If
```

Этот же синтаксис можно использовать для введения кода процедуры `GreetMe`.

```
Sub GreetMe5()
    If Time < 0.5 Then
        MsgBox "Доброе утро"
    Elseif Time >=0.5 And Time < 0.75 Then
```

```
        MsgBox "Добрый день"
    Else
        MsgBox "Добрый вечер"
    End If
End Sub
```

Если в представленном синтаксисе условие выполняется, то выполняются соответствующие операторы, после чего конструкция `If-Then` заканчивается. Другими словами, при этом не оцениваются дополнительные условия. Такой синтаксис наиболее эффективен, однако многим эта программа может показаться сложной для понимания.

Далее представлен еще один способ реализовать рассматриваемый пример. В нем используются вложенные конструкции `If-Then-Else` (без `ElseIf`). Данная процедура также эффективна, и ее легко понять. Обратите внимание, что для каждого оператора `If` существует свой оператор `End If`.

```
Sub GreetMe6()
    If Time < 0.5 Then
        MsgBox "Доброе утро"
    Else
        If Time >=0.5 And Time < 0.75 Then
            MsgBox "Добрый день"
        Else
            If Time >=0.75 Then
                MsgBox "Добрый вечер"
            End If
        End If
    End If
End Sub
```

Ниже продемонстрирован еще один пример, использующий простую форму конструкции `If-Then`. Процедура запрашивает у пользователя значение переменной `Quantity` и отображает скидку на основе полученного значения. Если в окне `InputBox` пользователь щелкнет на кнопке Отмена, то переменная `Quantity` приравняется пустой строке, в результате процедура заканчивается. Следует отметить, что эта процедура не выполняет других проверок (например, мы не проверяем в данном случае введенное числовое значение на отрицательность).

```
Sub Discount1()
    Quantity = InputBox("Введите значение: ")
    If Quantity = "" Then Exit Sub
    If Quantity >= 0 Then Discount = 0.1
    If Quantity >= 25 Then Discount = 0.15
    If Quantity >= 50 Then Discount = 0.2
    If Quantity >= 75 Then Discount = 0.25
    MsgBox "Скидка: " & Discount
End Sub
```

Обратите внимание, что каждый оператор `If-Then` в представленной процедуре всегда выполняется, а значение `Discount` может изменяться. Однако в результате всегда отображается нужное значение.

Следующая процедура — это вариант предыдущей, переписанной с использованием альтернативного синтаксиса. Процедура будет окончена после выполнения блока *операторы_истина*.

```
Sub Discount2()
    Quantity = InputBox("Введите значение: ")
    If Quantity = "" Then Exit Sub

    If Quantity >= 0 And Quantity < 25 Then
        Discount = 0.1
    ElseIf Quantity < 50 Then
```

```
        Discount = 0.15
    ElseIf Quantity < 75 Then
        Discount = 0.2
    ElseIf Quantity >= 75 Then
        Discount = 0.25
    End If
    MsgBox "Скидка: " & Discount
End Sub
```

Вложенные структуры `If-Then` достаточно громоздкие. Поэтому рекомендуется использовать их только для принятия простых бинарных решений. Если же необходимо выбрать между тремя и более вариантами, то целесообразно обратиться к конструкции `Select Case`.

Конструкции Select Case

Конструкция `Select Case` применяется при выборе между тремя и более вариантами. Она справедлива также для двух вариантов и является хорошей альтернативой структуре `If-Then-Else`. Конструкция `Select Case` имеет следующий синтаксис.

```
Select Case тестируемое_выражение
    [Case список условий-n
        [операторы-n]]
    [Case Else
        [операторы_по_умолчанию]]
End Select
```

Функция VBA IIf

VBA предлагает альтернативу конструкции `If-Then` — функцию `IIf`. Эта функция имеет три аргумента и работает подобно функции ЕСЛИ Excel. Рассмотрим ее синтаксис.

`IIf(выражение, часть_True, часть_False)`, где

выражение (обязательный аргумент) — выражение, которое необходимо оценить;

часть_True (обязательный аргумент) — значение или выражение, которое возвращает функция, если *выражение* имеет значение `True`;

часть_False (обязательный аргумент) — значение или выражение, которое возвращает функция, если *выражение* имеет значение `False`.

Следующая инструкция демонстрирует применение функции `IIf`. В окне сообщений отображается *Ноль*, если ячейка `A1` содержит ноль или пуста, и *Не-ноль*, если ячейка `A1` содержит другое значение.

```
MsgBox IIf (Range("A1") = 0, "Ноль", "Не-ноль")
```

Важно понимать, что третий аргумент (*часть_False*) всегда оценивается, даже если второй аргумент (*часть_True*) выполняется. Следовательно, следующий оператор выдаст ошибку, если значение `n` равно нулю.

```
MsgBox IIf (n = 0, 0, 1/n)
```

Приведем пример конструкции `Select Case`, который показывает еще один способ запрограммировать процедуру `GreetMe`, рассмотренную в предыдущем разделе.

```
Sub GreetMe()
    Select Case Time
        Case Is < 0.5
            Msg = "Доброе утро"
        Case 0.5 To 0.75
            Msg = "Добрый день"
        Case Else
            Msg = "Добрый вечер"
    End Select
    MsgBox Msg
End Sub
```

Ниже приведена версия процедуры Discount, переписанная с использованием конструкции Select Case. Эта процедура предполагает, что Quantity всегда выражается целым числом. Чтобы упростить процедуру, в ней не выполняется проверка введенных данных.

```
Sub Discount3()
    Quantity = InputBox("Введите значение: ")
    Select Case Quantity
        Case ""
            Exit Sub
        Case 0 To 24
            Discount = 0.1
        Case 25 To 49
            Discount = 0.15
        Case 50 To 74
            Discount = 0.2
        Case Is >= 75
            Discount = 0.25
    End Select
    MsgBox "Скидка: " & Discount
End Sub
```

В операторе Case может также использоваться оператор Or. В следующей процедуре для определения, каким днем является текущий (субботой или воскресеньем), применена функция VBA WeekDay, после чего отображается соответствующее сообщение. Обратите внимание на использование оператора Or, который проверяет день недели на принадлежность к субботе или воскресенью.

```
Sub GreetUser()
    Select Case Weekday(Now)
        Case 1 Or 7
            MsgBox "Это выходные."
        Case Else
            MsgBox "Это не выходные."
    End Select
End Sub
```

Под каждым оператором Case может указываться любое количество инструкций, и они выполняются при условии Case, имеющем значение True. Если вы используете в каждом случае Case только одну инструкцию (как в предыдущем примере), то можете поместить ее в той же строке, что и ключевое слово Case (но не забудьте про символ разделения операторов в VBA — двоеточие). Этот прием сделает текст программы более компактным.

```
Sub Discount3()
    Quantity = InputBox("Введите количество: ")
    Select Case Quantity
        Case "": Exit Sub
        Case 0 To 24: Discount = 0.1
        Case 25 To 49: Discount = 0.15
```

```
        Case 50 To 74: Discount = 0.2
        Case Is >= 75: Discount = 0.25
    End Select
    MsgBox "Скидка: " & Discount
End Sub
```

> **СОВЕТ** VBA выходит из конструкции `Select Case`, как только найдено условие `Case`, которое имеет значение `True`. Следовательно, для максимальной эффективности в первую очередь следует выполнять проверку наиболее вероятного случая.

Структуры `Select Case` можно вкладывать друг в друга. Например, следующая процедура выполняет проверку состояния окна Excel (развернутое, свернутое, нормальное), а затем отображает сообщение с описанием состояния. Если окно Excel находится в нормальном состоянии, то процедура проверяет состояние активного окна и отображает еще одно сообщение.

```
Sub AppWindow()
    Select Case Application.WindowState
        Case xlMaximized: MsgBox "Окно приложения развернуто"
        Case xlMinimized: MsgBox "Окно приложения свернуто"
        Case xlNormal: MsgBox "Окно приложения в нормальном состоянии"
            Select Case ActiveWindow.WindowState
                Case xlMaximized: MsgBox "Окно книги развернуто"
                Case xlMinimized: MsgBox "Окно книги свернуто"
                Case xlNormal: MsgBox "Окно книги в нормальном состоянии"
            End Select
    End Select
End Sub
```

Вы можете создавать конструкции `Select Case` любой степени вложенности, но убедитесь, что каждому оператору `Select Case` соответствует свой оператор `End Select`.

Данная процедура демонстрирует, насколько важно использовать отступы в коде, чтобы выделить его структуру. Например, рассмотрим ту же процедуру, но без отступов.

```
Sub AppWindow()
Select Case Application.WindowState
Case xlMaximized: MsgBox "Окно приложения развернуто"
Case xlMinimized: MsgBox "Окно приложения свернуто"
Case xlNormal: MsgBox "Окно приложения в нормальном состоянии"
Select Case ActiveWindow.WindowState
Case xlMaximized: MsgBox "Окно книги развернуто"
Case xlMinimized: MsgBox "Окно книги свернуто"
Case xlNormal: MsgBox "Окно книги в нормальном состоянии"
End Select
End Select
End Sub
```

В таком представлении мало что понятно даже опытному программисту!

Циклическая обработка инструкций

Цикл — это процесс повторения набора инструкций. Возможно, вы заранее знаете, сколько раз должен повториться цикл, или это значение определяется переменными в программе.

Следующий код, в котором в диапазон вводятся последовательные числа, является примером того, что называют *плохим циклом*. Процедура использует две переменные для хранения начального значения (`StartVal`) и общего количества ячеек, которые необходимо заполнить (`NumToFill`). В этом цикле для управления порядком выполнения операций используется оператор `GoTo`. Если переменная `Cnt`, отвечающая за то, сколько ячеек заполнено, меньше числа, заданного пользователем, то выполняется переход назад к метке `DoAnother`.

```
Sub BadLoop()
    StartVal = 1
    NumToFill = 100
    ActiveCell.Value = StartVal
    Cnt = 1
DoAnother:
    ActiveCell.Offset(Cnt,0).Value = StartVal + Cnt
    Cnt = Cnt + 1
    If Cnt < NumToFill Then GoTo DoAnother Else Exit Sub
End Sub
```

Описанная процедура выполняется правильно. Почему же она является примером плохого цикла? Настоящие программисты не используют оператор GoTo, если можно обойтись без него. Применение операторов GoTo в цикле противоречит концепции структурированного программирования. Этот оператор значительно усложняет восприятие кода, поскольку в данном случае практически невозможно структурировать цикл с помощью отступов. Кроме того, такой тип неструктурированного цикла делает процедуру подверженной частым ошибкам. Также использование большого количества меток приводит к получению программы с плохой структурой (или без структуры вообще) и бессистемным порядком следования операций.

Поскольку в VBA встроено несколько структурированных команд циклов, в процессе принятия решений практически никогда не требуется прибегать к операторам GoTo.

Несколько слов о структурном программировании

Пообщайтесь с профессиональными программистами, и рано или поздно вы услышите термин *структурное программирование*. Кроме того, вы обнаружите, что структурные программы ценятся выше, чем программы неструктурированные.

Что же представляет собой структурированное программирование и как его можно применить к VBA?

Основное условие заключается в том, что процедура или сегмент программы должны иметь только одну точку входа и одну точку выхода. Другими словами, тело кода должно выступать независимым элементом, а контроль за выполнением программы не должен выходить за рамки этого элемента. В результате, структурное программирование не приемлет оператора GoTo. Если вы обеспечиваете структурированность кода, то ваша программа выполняется в определенном порядке, и за ходом выполнения легко уследить — в отличие от кода, в котором программа "перескакивает" с места на место.

Структурированную программу легче воспринимать и анализировать, чем неструктурированную. Что более важно, ее также легче изменять.

VBA — структурированный язык. Он предлагает стандартные структурные конструкции, например, If-Then-Else и Select Case, циклы For-Next, Do Until и Do While. Более того, VBA полностью поддерживает модульную систему создания программ.

Если вы начинающий программист, то можете сразу приступить к выработке хорошего стиля программирования.

Циклы For-Next

Простейший пример *хорошего цикла* — For-Next (он уже применялся в нескольких предыдущих примерах). Этот оператор имеет следующий синтаксис.

```
For счетчик = начало To конец [Step шаг]
    [инструкции]
    [Exit For]
    [инструкции]
Next [счетчик]
```

Ниже приведен пример цикла `For-Next`, в котором не используется переменная *шаг* и необязательный оператор `Exit For`. Эта процедура выполняет оператор `Sum = Sum + Sqr(Count)` 100 раз и отображает результат — сумму квадратных корней первых 100 целых чисел.

```
Sub SumSquareRoots()
    Dim Sum As Double, Count As Integer
    Sum = 0
    For Count = 1 To 100
        Sum = Sum + Sqr(Count)
    Next Count
    MsgBox Sum
End Sub
```

В данном примере `Count` (переменная-счетчик цикла) имеет начальное значение 1 и увеличивается на 1 при каждом повторении цикла. Переменная `Sum` суммирует квадратные корни каждого значения `Count`.



Используя циклы `For-Next`, важно понимать, что счетчик цикла является обычной переменной, и ничем более. В результате значение счетчика цикла можно изменять в блоке программы, который выполняется между операторами `For` и `Next`. Однако это *очень неудачный* прием, так как он может вызвать непредсказуемые результаты. Необходимо принять специальные меры предосторожности, чтобы удостовериться, что программа не изменяет счетчик цикла.

Вы можете также использовать переменную *шаг*, чтобы пропустить отдельные итерации цикла. Ниже рассмотрена исходная процедура, переписанная так, что суммируются квадратные корни всех нечетных чисел в диапазоне от 1 до 100.

```
Sub SumOddSquareRoots()
    Sum = 0
    For Count = 1 To 100 Step 2
        Sum = Sum + Sqr(Count)
    Next Count
    MsgBox Sum
End Sub
```

В этой процедуре исходное значение `Count` равно 1, затем счетчик принимает значения 3, 5, 7 и т.д. Последнее значение `Count`, используемое в цикле, равно 99. Когда цикл заканчивается, значение `Count` равно 101.

Представленная далее процедура выполняет ту же задачу, что и `BadLoop`, описанный в начале раздела “ Циклическая обработка инструкций”. Однако в данном случае не используется оператор `GoTo`, поэтому неудачный цикл превращается в удачный. В нем задействована структура `For-Next`.

```
Sub GoodLoop()
    StartVal = 1
    NumToFill = 100
    ActiveCell.Value = StartVal
    For Cnt = 0 To NumToFill - 1
      ActiveCell.Offset(Cnt,0).Value = StartVal + Cnt
    Next Cnt
End Sub
```

Циклы `For-Next` могут также содержать один или более операторов `Exit For`. Когда программа встречает этот оператор, то сразу же выходит из цикла, как показано в следующем примере. Эта процедура определяет, какая ячейка столбца A на активном рабочем листе имеет наибольшее значение.

```
Sub ExitForDemo()
    MaxVal = Application.WorksheetFunction.Max(Range("A:A"))
    For Row = 1 To 65536
        Set TheCell = Range("A1").Offset(Row - 1, 0)
        If TheCell.Value = MaxVal Then
            MsgBox "Максимальное значение находится в строке " & Row
            TheCell.Activate
            Exit For
        End If
    Next Row
End Sub
```

Максимальное значение в столбце вычисляется с помощью функции Excel `MAX`. Затем это значение присваивается переменной `MaxVal`. Цикл `For-Next` проверяет каждую ячейку в столбце. Если определенная ячейка равна `MaxVal`, оператор `Exit For` заканчивает процедуру. Однако перед выходом из цикла процедура сообщает пользователю о расположении искомой ячейки и активизирует ее.

Пример `ExitForDemo` представлен с целью продемонстрировать выход из цикла `For-Next`. Однако это не самый эффективный способ найти максимальное значение в диапазоне. Поставленную задачу может выполнить единственный оператор.

```
Range("A:A").Find(Application.WorksheetFunction.Max _
    (Range("A:A"))).Activate
```

В предыдущих примерах использовались достаточно простые циклы. Но вы можете помещать в цикл любое количество операторов и даже вкладывать циклы `For-Next` в другие циклы `For-Next`. Ниже приведен пример, в котором применены вложенные циклы `For-Next` для инициализации массива 10×10×10 значением -1. По завершении выполнения процедуры все 1000 элементов массива `MyArray` будут содержать значение -1.

```
Sub NestedLoops()
    Dim MyArray(1 To 10, 1 To 10, 1 To 10)
    Dim i As Integer, j As Integer, k As Integer
    For i = 1 To 10
        For j = 1 To 10
            For k = 1 To 10
                MyArray(i, j, k) = -1
            Next k
        Next j
    Next i
End Sub
```

ЦИКЛЫ DO WHILE

Оператор `Do While` — еще один тип циклической структуры, представленной в VBA. В отличие от цикла `For-Next`, цикл `Do While` выполняется до тех пор, пока удовлетворяется заданное условие. Цикл `Do While` может иметь один из двух представленных ниже синтаксисов.

```
Do [While условие]
    [инструкции]
    [Exit Do]
    [инструкции]
Loop
```

или

```
Do
    [инструкции]
    [Exit Do]
    [инструкции]
Loop [While условие]
```

Как видите, VBA позволяет проверять условие `While` в начале или в конце цикла. Разница между этими двумя синтаксисами связана с моментом, когда оценивается условие. В первом синтаксисе содержимое цикла может вообще не выполняться. Во втором содержимое цикла всегда выполняется (как минимум один раз).

Следующий пример демонстрирует цикл `Do While` с первым синтаксисом.

```
Sub DoWhileDemo()
    Do While Not IsEmpty(ActiveCell)
        ActiveCell.Value = 0
        ActiveCell.OffSet(1, 0).Select
    Loop
End Sub
```

Данная процедура использует активную ячейку как точку отсчета и просматривает значения вниз по столбцу, вставляя ноль в активную ячейку. При каждом повторении цикла активной становится следующая ячейка в столбце. Цикл продолжается, пока функция VBA `IsEmpty` не определит, что активная ячейка пуста.

Далее рассмотрим работу второго варианта синтаксиса цикла `Do While`. Цикл всегда выполняется хотя бы один раз, даже если исходная активная ячейка пуста.

```
Sub DoWhileDemo2()
    Do
        ActiveCell.Value = 0
        ActiveCell.OffSet(1, 0).Select
    Loop While Not IsEmpty(ActiveCell)
End Sub
```

Ниже следует еще один пример цикла `Do While`. Эта процедура открывает текстовый файл, считывает каждую строку, преобразует текст в верхний регистр, а затем сохраняет его на активном листе, начиная с ячейки `A1`, и продолжает перемещаться вниз по столбцу. Представленная процедура использует функцию VBA `EOF`, возвращающую `True`, если достигнут конец файла. Последний оператор закрывает текстовый файл.

```
Sub DoWhileDemo1()
    Open "c:\data\textfile.txt" For Input As #1
    LineCt = 0
    Do While Not EOF(1)
      Line Input #1, LineOfText
      Range("A1").Offset(LineCt, 0) = UCase(LineOfText)
      LineCt = LineCt + 1
    Loop
    Close #1
End Sub
```

ПЕРЕКРЕСТНАЯ ССЫЛКА

Дополнительную информацию о чтении и записи текстовых файлов с использованием VBA вы найдете в главе 27.

Циклы Do Until

Структура цикла `Do Until` имеет много общего с конструкцией `Do While`. Разница заключается лишь в том, как проверяется условие цикла. В варианте `Do While` цикл выполняется *до тех пор, пока* выполняется условие. В цикле `Do Until` цикл выполняется, *пока* условие не станет выполняться.

Структура `Do Until` может быть представлена двумя вариантами синтаксиса.

```
Do [Until условие]
    [инструкции]
    [Exit Do]
    [инструкции]
Loop
```

или

```
Do
    [инструкции]
    [Exit Do]
    [инструкции]
Loop [Until условие]
```

Пример, приводимый далее, уже был продемонстрирован для цикла `Do While`, но теперь он изменен для иллюстрации возможностей цикла `Do Until`. Единственное отличие — строка с оператором `Do`. Этот пример делает программу несколько понятнее, так как не используется отрицание, необходимое в примере `Do While`.

```
Sub DoUntilDemo1()
    Open "c:\data\textfile.txt" For Input As #1
    LineCt = 0
    Do Until EOF(1)
      Line Input #1, LineOfText
      Range("A1").Offset(LineCt, 0) = UCase(LineOfText)
      LineCt = LineCt + 1
    Loop
    Close #1
End Sub
```

Глава 9

Работа с процедурами VBA

В ЭТОЙ ГЛАВЕ...

Процедура содержит группу операторов VBA, которые выполняют поставленную задачу. Большая часть кода VBA содержится в процедурах. Данная глава сосредоточена на *процедурах*, которые выполняют задачи, но не возвращают дискретные значения.

- Объявление и создание процедур VBA
- Выполнение процедур
- Передача аргументов в процедуру
- Методы обработки ошибок
- Пример полезной процедуры

ПЕРЕКРЕСТНАЯ ССЫЛКА

В главе 11 приведены примеры процедур, которые вы можете использовать в своей работе.

О процедурах

Процедура — это последовательность операторов VBA, расположенная в модуле VBA, доступ к которому вы получаете с помощью VBE. Модуль может включать любое количество процедур.

Существует несколько способов *вызвать*, или выполнить, процедуры. Процедура выполняется от начала до конца (этот процесс также можно преждевременно прервать).

СОВЕТ

Процедура может иметь любую длину, но многие предпочитают избегать чересчур длинных процедур, выполняющих большое количество разных операций. Возможно, проще будет написать несколько меньших процедур, каждая из которых выполняет свою задачу, а затем разработать главную процедуру, вызывающую эти маленькие процедуры. Такой подход облегчит дальнейшее управление вашей программой.

Некоторые процедуры получают аргументы. *Аргумент* — это информация, используемая процедурой в процессе выполнения. Аргументы процедуры во многом подобны аргументам, используемым функциями Excel. Инструкции в процедуре обычно выполняют логические операции над аргументами, а результаты процедуры обычно основаны на представляемых ей аргументах.

Объявление процедуры

При объявлении процедуры с использованием ключевого слова `Sub` необходимо придерживаться следующего синтаксиса.

```
[Private | Public][Static] Sub имя ([список_аргументов])
  [инструкции]
  [Exit Sub]
  [инструкции]
End Sub
```

- `Private` (необязательное ключевое слово) — указывает на то, что процедура доступна только для других процедур в том же модуле.
- `Public` (необязательное ключевое слово) — указывает на то, что процедура доступна для всех остальных процедур во всех модулях рабочей книги. При использовании в модуле, содержащем оператор `Option Private Module`, процедура будет не доступна за пределами проекта.
- `Static` (необязательное ключевое слово) — указывает на то, что переменные процедуры сохраняются после окончания процедуры.
- `Sub` (обязательное ключевое слово) — обозначает начало процедуры.
- *`имя`* — любое корректное название процедуры.
- *`список_аргументов`* — представляет заключенный в скобки список переменных, содержащих аргументы, которые передаются в процедуру. Для разделения аргументов используется запятая. Если процедура не использует аргументы, то необходимо включить в объявление процедуры пустые скобки.
- *`инструкции`* (необязательные) — корректные инструкции VBA.
- `Exit Sub` (необязательный оператор) — вызывает немедленный выход из процедуры до ее формального завершения.
- `End Sub` (обязательный оператор) — указывает на окончание процедуры.

НА ЗАМЕТКУ

За некоторым исключением, все инструкции VBA в модуле должны содержаться в процедурах. Эти исключения касаются объявления переменных уровня модуля, определения пользовательских типов данных и некоторых других инструкций, определяющих параметры на уровне модуля (например, `Option Explicit`).

Название процедуры

Каждая процедура должна иметь название. Правила именования процедур, в основном, такие же, как при именовании переменных. В идеале название процедуры должно описывать, что выполняют операторы, содержащиеся в процедуре. Хорошее правило — использование названия, включающего глагол и существительное (например, `ProcessDate` — обработать данные, `PrintReport` – распечатать отчет, `Sort_Array` – сортировать массив или `CheckFilename` – проверить имя файла). Избегайте незначимых названий (`DoIt`, `Update`, `Fix` и т.п.).

Некоторые программисты используют названия, которые структурно напоминают предложения, описывающие процедуру (например, `WriteReportToTextFile` – записать отчет в текстовый файл, `Get_Print_Options_and_Print_Report` – получить параметры печати и распечатать отчет). Длинные названия точны и однозначно представляют предназначение процедуры, однако вводить их намного сложнее и дольше.

Область действия процедуры

В предыдущей главе отмечалось, что область действия переменной определяется модулями и процедурами, в которых может использоваться переменная. Аналогичным образом область действия процедуры определяет, какие процедуры могут ее вызывать.

ПРОЦЕДУРЫ ТИПА PUBLIC

По умолчанию все процедуры имеют область действия `Public`, т.е. они могут быть вызваны другими процедурами в любом модуле рабочей книги. Ключевое слово `Public` использовать необязательно, но программисты часто включают его для ясности. Обе приведенные ниже процедуры имеют область действия `Public`.

```
Sub First()
'  ...[код процедуры]..
End Sub

Public Sub Second()
'  ...[код процедуры]..
End Sub
```

ПРОЦЕДУРЫ ТИПА PRIVATE

Процедуры типа `Private` могут быть вызваны другими процедурами в этом же модуле, но не процедурами других модулей.

При выборе команды Сервис⇨Макрос⇨Макросы в Excel в диалоговом окне Макрос отображаются только `Public`-процедуры. Следовательно, при наличии процедур, которые должны вызываться только процедурами в этом же модуле, необходимо убедиться, что процедура имеет область действия `Private` — таким образом вы предотвратите ее запуск из диалогового окна Макрос.

В следующем примере объявляется процедура типа `Private` с названием `MySub`.

```
Private Sub MySub()
'  ...[код процедуры]
End Sub
```

Вы можете назначить всем процедурам модуля область действия `Private` — даже тем, которые объявлены с использованием ключевого слова `Public`. Для этого включите следующий оператор перед первым оператором `Sub`.

```
Option Private Module
```

Введя указанный оператор в модуле, вы можете опустить ключевое слово `Private` в объявлениях процедур.

Функция записи макросов Excel обычно создает новые процедуры с названием `Макрос1`, `Макрос2` и т.д. Все эти процедуры имеют область действия `Public` и не имеют аргументов.

Выполнение процедуры

Далее представлены некоторые способы *выполнения*, или вызова, процедуры VBA.

- Используйте команду Run⇨Run Sub/UserForm (в VBE). Альтернатива — нажать <F5>. Excel выполняет процедуру, в которой находится курсор. Этот метод не срабатывает, если процедура имеет один или более аргументов.

- Из диалогового окна Макрос программы Excel (которое можно открыть, выбрав команду Сервис⇨Макрос⇨Макросы). Другой способ отобразить диалоговое окно Макрос — нажать <Alt+F8>.
- С помощью комбинации клавиши <Ctrl> и присвоенной процедуре клавиши (если процедуре присвоена комбинация клавиш).
- Щелкнув на кнопке или форме рабочего листа. Для этого кнопке или форме должна быть присвоена процедура.
- Из другой процедуры.
- С помощью кнопки на панели инструментов.
- Из специально разработанного меню.
- По выполнению определенного события. Такими событиями может выступать открытие рабочей книги, сохранение рабочей книги, закрытие рабочей книги, изменение ячейки, переход на другой рабочий лист и многое другое.
- Из окна Immediate редактора Visual Basic. Просто введите название процедуры, включите необходимые аргументы и нажмите <Enter>.

Рассмотренные методы запуска процедур подробно обсуждаются в следующих разделах.



Во многих случаях процедура не будет правильно работать, если она не выполняется в правильном контексте. Например, если процедура должна работать с активным рабочим листом, она выдаст ошибку, когда активным является лист диаграммы. Хорошая процедура включает код, выполняющий проверку соответствующего контекста, и вместо сообщения об ошибке элегантно извещает о завершении своей работы.

Выполнение процедуры с помощью команды Run⇨Run Sub/UserForm

Команда меню Run⇨Run Sub/UserForm в VBE используется преимущественно для тестирования процедуры в процессе ее разработки. Пользователь вряд ли станет активизировать редактор Visual Basic, чтобы запустить процедуру. Выберите команду Run⇨Run Sub/UserForm (или нажмите клавишу <F5>) в VBE, чтобы выполнить текущую процедуру (другими словами, процедуру, в которой расположен курсор).

Если курсор во время выбора команды Run⇨Run Sub/UserForm находится не в одной из процедур, то VBE отображает диалоговое окно Макрос, в котором можно выбрать процедуру для выполнения.

Выполнение процедуры в диалоговом окне Макрос

При выборе команды Сервис⇨Макрос⇨Макросы в Excel отображается диалоговое окно Макрос — рис. 9.1 (кроме того, это диалоговое окно можно открыть с помощью комбинации клавиш <Alt+F8>). В диалоговом окне Макрос перечислены все созданные вами процедуры. Используйте раскрывающийся список Находится в, чтобы отфильтровать отображаемые макросы по области действия (например, показать только макросы активной рабочей книги). В диалоговом окне Макрос не представлены процедуры функций. В нем также вы не увидите процедуры, объявленные с ключевым словом `Private`, процедуры, требующие аргументов, или процедуры надстроек.

СОВЕТ Процедуры, которые хранятся в надстройках, не перечислены в диалоговом окне Макрос, но их можно выполнить, если вы знаете точное название процедуры. Введите название в поле Имя макроса диалогового окна Макрос и щелкните на кнопке Выполнить.

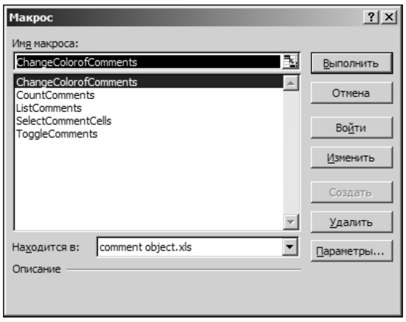


Рис. 9.1. В диалоговом окне Макрос *перечислены все доступные процедуры*

Выполнение процедуры с помощью комбинации клавиш

Любой процедуре, не имеющей аргументов, можно присвоить комбинацию <Ctrl+специальная клавиша>. Например, если вы присвоите процедуре с названием `Update` комбинацию клавиш <Ctrl+U>, то при нажатии <Ctrl+U> она будет выполняться.

В начале записи макроса в диалоговом окне Запись макроса вам предоставляется возможность задать комбинацию клавиш, которая будет использоваться для его выполнения. Ее можно присвоить в любое удобное для вас время. Чтобы присвоить процедуре комбинацию клавиш (или изменить уже существующую), выполните следующие действия.

1. Запустите Excel и выберите команду Сервис⇨Макрос⇨Макросы.
2. Выберите необходимую процедуру из списка в диалоговом окне Макрос.
3. Щелкните на кнопке Параметры, чтобы отобразить диалоговое окно Параметры макроса, показанное на рис. 9.2.
4. Введите символ в текстовое поле Ctrl+.
5. Символ, который вы вводите в текстовое поле Ctrl+, чувствителен к регистру. Если вы введете *s* в нижнем регистре, то будет присвоена комбинация клавиш <Ctrl+S>. Если вы введете *S* в верхнем регистре, то будет присвоена комбинация клавиш <Ctrl+Shift+S>.

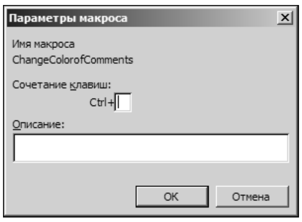


Рис. 9.2. В диалоговом окне Параметры макроса *можно присвоить процедуре комбинацию клавиш и (не обязательно) описание*

6. Добавьте описание (по желанию). Если описание введено, оно будет отображаться в нижней части диалогового окна Макрос при выборе процедуры из списка.
7. Щелкните на кнопке OK, чтобы закрыть диалоговое окно Параметры макроса, затем щелкните на кнопке Закрыть, чтобы закрыть диалоговое окно Макрос.

Если вы присваиваете процедуре одну из предопределенных комбинаций клавиш Excel, то ваша команда имеет приоритет над предопределенной комбинацией клавиш. Например, комбинация клавиш <Ctrl+S> в Excel соответствует операции сохранения рабочей книги. Но если вы присвоите эту же комбинацию процедуре, то при нажатии <Ctrl+S> активная книга больше сохраняться не будет.

В Excel не используются следующие комбинации <Ctrl+*клавиша*>: E, J, L, M, Q и T. В Excel практически не применяются комбинации клавиш <Ctrl+Shift+*клавиша*>, поэтому свободно используйте любые комбинации, *кроме* F, O и P.

Выполнение процедуры из пользовательского меню

Как описано в главе 23, Excel предоставляет два способа создания пользовательских меню: с помощью команды Вид⇨Панели инструментов⇨Настройка и в результате создания программы VBA. Предпочтительнее использовать второй метод, но для связывания макроса с новым элементом меню можно применять оба метода.

Ниже приведены действия, которые следует выполнить для отображения в меню нового элемента и связывания этого элемента с макросом. Предполагается, что новый элемент добавляется в меню Данные: название нового элемента — Открыть файл заказчика; процедура называется `OpenCustomerFile`.

1. Выберите команду Вид⇨Панели инструментов⇨Настройка. Excel отображает диалоговое окно Настройка.
2. При отображении диалогового окна Настройка программа переходит в специальный режим "настройки". Меню и панели инструментов неактивны, а их параметры можно изменять.
3. Щелкните на вкладке Команды в диалоговом окне Настройка.
4. Прокрутите вниз список Категории и выберите опцию Макросы.
5. Перетащите из списка Команды первый элемент с названием Настраиваемая команда меню в нижнюю часть меню Данные. Расположите его после элемента Обновить данные. Меню Данные будет отображено при щелчке на нем.
6. Щелкните правой кнопкой мыши на новом элементе меню (с названием Настраиваемая команда меню), чтобы отобразить контекстное меню.
7. Введите в текстовое поле Имя новое название элемента меню: **&Открыть файл заказчика**, как показано на рис. 9.3.
8. Щелкните на команде контекстного меню Назначить макрос.
9. В диалоговом окне Назначить макрос выберите процедуру `OpenCustomerFile` из списка макросов.
10. Щелкните на кнопке OK, чтобы закрыть диалоговое окно Назначить макрос, а затем — на кнопке Закрыть, чтобы закрыть диалоговое окно Настройка.

После выполнения указанных выше действий новый элемент всегда будет отображаться в меню, даже если рабочая книга, содержащая макрос, закрыта. Другими словами, изменения, проведенные с помощью команды Вид⇨Панели инструментов⇨Настройка, являются *постоянными*. При выборе нового элемента меню рабочая книга, если она еще не открыта, будет открыта.

В главе 23 рассказано об использовании VBA для создания элементов меню, которые отображаются, если открыта только определенная рабочая книга.

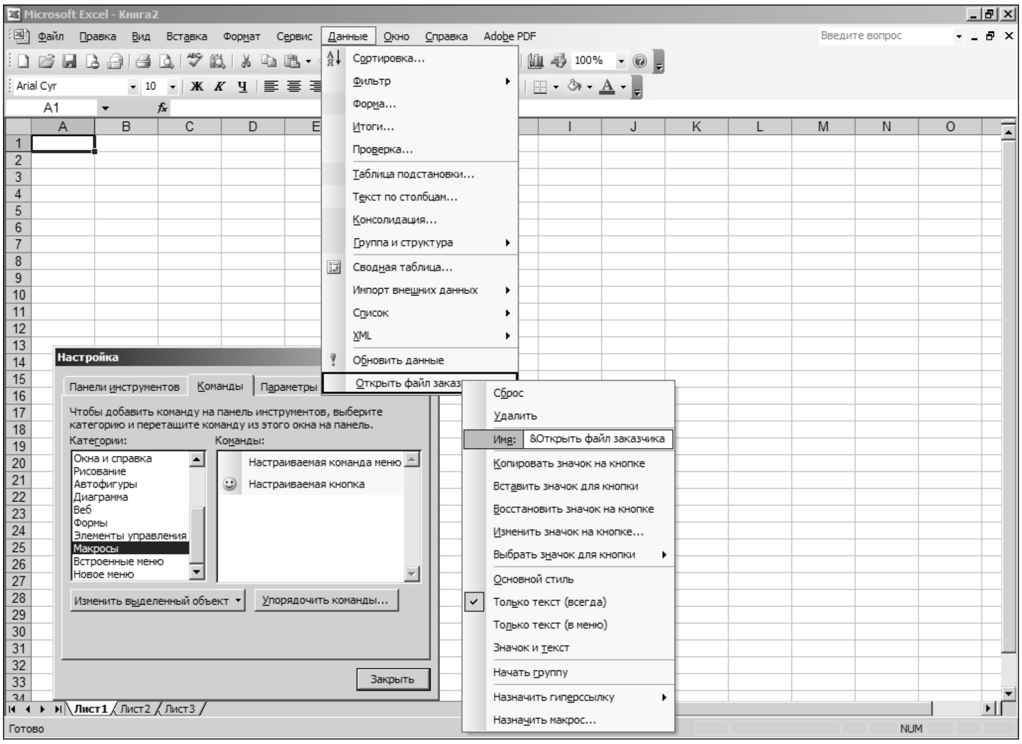


Рис. 9.3. Изменение названия элемента меню

Выполнение процедуры из другой процедуры

Один из самых популярных способов выполнить процедуру — вызвать ее из другой процедуры. Для этого существует три способа.

- Ввести название процедуры и необходимые аргументы этой процедуры (если они есть) через запятую.
- Ввести ключевое слово `Call`, после него — название процедуры и ее аргументы (если они есть), заключенные в скобки и разделенные запятыми.
- Использовать метод `Run` объекта `Application`. Этот метод можно применять для выполнения других процедур VBA или макросов XLM. Метод `Run` полезен, когда выполняется процедура, название которой присвоено переменной. Тогда переменную можно передать в метод `Run` как аргумент.

В следующем примере показан первый метод. В данном случае процедура `MySub` выполняет некоторые операторы (не показанные в примере), запускает процедуру `UpdateSheet` и затем выполняет остальные операторы.

```
Sub MySub()
'  ...[код]...
  UpdateSheet
'  ...[код]...
End Sub
```

```
Sub UpdateSheet()
'  ...[код]...
End Sub
```

Далее представлен второй метод. Ключевое слово `Call` вызывает процедуру `Update`, имеющую один аргумент; вызывающая процедура передает аргумент в вызываемую процедуру. Аргументы процедур рассматриваются далее в этой главе.

```
Sub MySub()
  MonthNum = InputBox("Введите номер месяца: ")
  Call UpdateSheet(MonthNum)
'  ...[код]...
End Sub

Sub UpdateSheet(MonthSeq)
'  ...[код]...
End Sub
```

СОВЕТ

Некоторые программисты всегда используют ключевое слово `Call`, чтобы явно указать на вызов другой процедуры (однако такое действие необязательно).

В следующем примере используется метод `Run`, выполняющий процедуру `UpdateSheet` и передающий `MonthNum` в качестве аргумента.

```
Sub MySub()
  MonthNum = InputBox("Введите номер месяца: ")
  Result = Application.Run("UpdateSheet", MonthNum)
'  ...[код]...
End Sub

Sub UpdateSheet(MonthSeq)
'  ...[код]...
End Sub
```

Возможно, главной причиной использования метода `Run` является присвоение названия процедуры переменной. Существует только один способ выполнить процедуру таким образом (см. следующий пример). Процедура `Main` использует функцию VBA `WeekDay` для определения дня недели (целое число от 1 до 7, начиная с воскресенья). Переменной `SubToCall` присваивается строка, содержащая название процедуры. Затем метод `Run` вызывает соответствующую процедуру (`Weekend` или `Daily`).

```
Sub Main()
  Select Case Weekday(Now)
    Case 1: SubToCall = "Weekend"
    Case 7: SubToCall = "Weekend"
    Case Else: SubToCall = "Daily"
  End Select
    Application.Run SubToCall
End Sub

Sub Weekend()
  MsgBox "Сегодня выходной"
'  Код, который выполняется в выходной
End Sub

Sub Daily()
  MsgBox "Сегодня будний день"
'  Код, который выполняется в будний день
End Sub
```

ВЫЗОВ ПРОЦЕДУРЫ ИЗ ДРУГОГО МОДУЛЯ

Если VBA не может найти вызываемую процедуру в текущем модуле, он ищет процедуры `Public` в других модулях этого же проекта.

При вызове процедуры `Private` из другой процедуры обе они должны находиться в одном модуле.

Следует отметить, что в одном модуле не может быть двух процедур с одинаковыми названиями, не допускается также использование процедуры с идентичными именами в разных модулях. Можно указать VBA выполнить *неоднозначно названную* процедуру, т.е. другую процедуру с тем же названием, но в другом модуле. Для этого введите перед названием процедуры название модуля и точку. Предположим, вы создали процедуры с названием `MySub` в двух модулях: `Module1` и `Module2`. Если требуется, чтобы процедура `MySub` в `Module2` вызывала `MySub` из `Module1`, то используйте один из следующих операторов.

```
Module1.MySub
Call Module1.MySub
```

Если вы не укажете разницу между процедурами с одинаковыми названиями, будет отображено сообщение об ошибке: `Ambiguous Name Detected` (Обнаружено неоднозначное имя).

ВЫЗОВ ПРОЦЕДУРЫ В ДРУГОЙ РАБОЧЕЙ КНИГЕ

В некоторых случаях необходимо, чтобы процедура выполняла иную процедуру, определенную в другой рабочей книге. Для этого существует два варианта: установить ссылку на другую рабочую книгу или использовать метод `Run` и явно указать название рабочей книги.

Чтобы добавить ссылку на другую рабочую книгу, выберите в VBE команду Tools⇨References. Будет отображено диалоговое окно References, как показано на рис. 9.4, в котором перечислены все существующие ссылки во всех открытых рабочих книгах. Выставьте флажок, соответствующий той рабочей книге, на которую вы планируете добавить ссылку, и щелкните на кнопке OK. После установки ссылки процедуры в этой рабочей книге можно вызывать так, будто они находятся в текущей рабочей книге, к которой принадлежит вызывающая процедура.

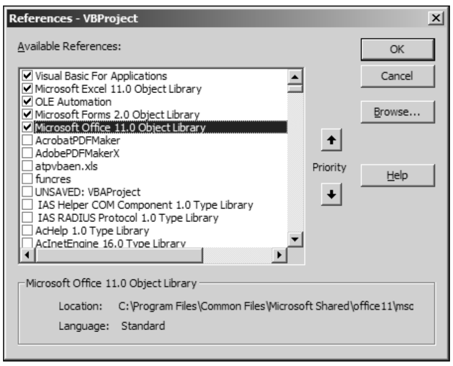


Рис. 9.4. В диалоговом окне References *можно установить ссылку на другую рабочую книгу*

Рабочая книга, на которую ссылаются, не обязательно должна быть открыта — она рассматривается как отдельная библиотека объектов. Используйте кнопку Browse в диалоговом окне References, чтобы установить ссылку на закрытую рабочую книгу. Названия книг, которые указаны в списке ссылок, перечислены по названиям проектов VBE. По умолчанию каждый проект обычно наделен именем *VBAProject*. Следовательно, в списке может содержаться несколько одинаково названных элементов. Чтобы отличить определенный проект от других элементов в списке, измените его название в окне Properties редактора VBE. Список ссылок, который отображается в диалоговом окне References, включает также библиотеки объектов и элементы управления ActiveX, зарегистрированные в вашей системе. Рабочие книги Excel 2003 всегда включают ссылки на библиотеки объектов:

- Visual Basic for Applications;
- Microsoft Excel 10.0 Object Library;
- OLE Automation;
- Microsoft Office 10.0 Object Library;
- Microsoft Forms 2.0 Object Library (необязательно; включается в случае, если в проекте присутствует пользовательская форма `UserForm`).

В Excel 2003 любые дополнительные ссылки, которые вы добавите, включаются в структуру проекта в окне Project Explorer редактора VBE. Эти ссылки перечислены в узле с названием References.

К примеру, если вы установили ссылку на рабочую книгу, которая содержит процедуру `YourSub`, то можете использовать один из следующих операторов для вызова `YourSub`.

```
YourSub
Call YourSub
```

Чтобы точно идентифицировать процедуру в другой рабочей книге, задайте имя проекта, название модуля и имя процедуры, придерживаясь такого синтаксиса.

```
MyProject.MyModule.MySub
```

Также можно использовать ключевое слово `Call`.

```
Call MyProject.MyModule.MySub
```

Существует еще один способ вызвать процедуру в другой открытой рабочей книге — использовать метод `Run` объекта `Application`. Этот метод не требует установки ссылки. Ниже представлен оператор, который выполняет процедуру `Consolidate`, расположенную в рабочей книге с названием `budget macros.xls`.

```
Application.Run "'budget macros.xls'!Consolidate"
```

Причины вызова других процедур

Если вы начинающий программист, то можете удивиться, зачем вообще необходимо вызывать процедуру из другой процедуры. Вы спросите: почему бы не поместить код из вызываемой процедуры в вызывающую и не усложнять жизнь?

Во-первых, это делает программу более понятной. Чем проще программа, тем легче ее анализировать и изменять. Меньшие процедуры можно быстрее понять и отладить. Рассмотрим

процедуру, которая только вызывает другие процедуры и больше не выполняет никаких функций. Вы разберетесь в ней достаточно быстро.

```
Sub Main()
  Call GetUserOptions
  Call ProcessData
  Call CleanUp
  Call CloseItDown
End Sub
```

Вызов других процедур также устраняет необходимость повторного введения кода. Предположим, необходимо выполнить определенную операцию в десяти местах программы. Вместо того чтобы вводить этот код десять раз, можно написать процедуру, выполняющую необходимую операцию, и затем десять раз вызвать процедуру.

Кроме того, вы будете часто работать с рядом процедур общего назначения. Если вы сохраните их в отдельном модуле, то сможете импортировать модуль в текущий проект и при необходимости вызывать эти процедуры, что значительно проще, чем копировать и вставлять код в новые процедуры.

Зачастую создание нескольких небольших процедур вместо одной большой считают хорошей практикой программирования. Модульный подход не только повышает эффективность работы, но и облегчает жизнь тем людям, которые впоследствии будут работать с вашей программой.

Выполнение процедуры с помощью кнопки на панели инструментов

Вы можете задать специальные настройки панелей инструментов Excel, чтобы добавить на них кнопку, щелчок на которой выполняет процедуру. Процедура присвоения макроса кнопке на панели инструментов практически идентична процедуре назначения макроса элементу меню.

Предположим, что вы решили присвоить процедуру кнопке на панели инструментов. Для этого выполните следующие действия.

1. Выберите команду Вид⇨Панели инструментов⇨Настройка, и вы увидите диалоговое окно Настройка.
2. При отображении диалогового окна Настройка Excel переходит в специальный режим "настройки". Меню и панели инструментов неактивны, а их параметры можно легко изменять.
3. Щелкните на вкладке Команды диалогового окна Настройка.
4. Прокрутите вниз список Категории и выберите опцию Макросы.
5. Перетащите из списка Команды второй элемент с названием Настраиваемая кнопка на требуемую панель инструментов.
6. Щелкните правой кнопкой мыши на новой кнопке, чтобы отобразить контекстное меню.
7. Введите в текстовое поле Имя название кнопки. Это текст "подсказки", которая появляется на экране, когда указатель мыши наведен на кнопку. Описанный шаг необязателен; если вы его пропустите, в подсказке будет отображаться Настраиваемая кнопка.
8. Щелкните правой кнопкой мыши на новой кнопке, чтобы отобразить контекстное меню, а затем выполните команду Назначить макрос. Excel отображает диалоговое окно Назначить макрос.

9. Выберите процедуру из списка макросов.
10. Щелкните на кнопке ОК, чтобы закрыть диалоговое окно Назначить макрос.
11. Щелкните на кнопке Закрыть, чтобы закрыть диалоговое окно Настройка.

После выполнения указанных выше действий новая кнопка всегда будет отображена на панели инструментов, даже если рабочая книга, содержащая макрос, закрыта. Другими словами, изменения, проведенные с помощью команды Вид⇨Панели инструментов⇨Настройка, являются *постоянными*. При использовании новой кнопки на панели инструментов рабочая книга, если она еще не открыта, открывается.

Пользовательские панели инструментов рассматриваются в главе 22.

Выполнение процедуры по щелчку на объекте

Excel содержит некоторые элементы управления, которые можно поместить на рабочем листе или листе диаграммы. Макрос допускается связывать с любым из них. Элементы управления доступны на трех панелях инструментов.

- Панели инструментов Рисование.
- Панели инструментов Формы.
- Панели инструментов Элементы управления.

Кроме того, макросы можно связывать с рисунками, добавляемыми на рабочий лист с помощью команды Вставка⇨Рисунок. Щелкните правой кнопкой на рисунке и выполните команду Назначить макрос.

Панель инструментов Элементы управления содержит те же элементы управления ActiveX, которые применяются в пользовательской форме `UserForm`. Панель инструментов Формы, включенная из соображений совместимости, содержит аналогичные элементы управления (не являющиеся элементами управления ActiveX). Элементы управления на панели инструментов Формы разработаны для Excel 5 и Excel 95. Однако их можно применять и в более поздних версиях (в некоторых случаях их использовать лучше). Изложенный далее материал относится к элементу управления Кнопка на панели инструментов Формы. Информацию о применении элементов управления ActiveX на рабочих листах вы найдете в главе 13.

Чтобы связать процедуру с объектом `Button` (Кнопка), который находится на панели инструментов Формы, выполните следующие действия.

1. Убедитесь, что на экране отображается панель инструментов Формы.
2. Щелкните на опции Кнопка на панели инструментов Формы.
3. Перетащите элемент Кнопка на рабочий лист.

Если при перетаскивании элемента нажать <Alt>, то кнопка будет выравниваться вдоль линий сетки рабочего листа. Если нажать <Shift>, то кнопка приобретет идеально квадратную форму.

Excel сразу же отображает окно Назначить макрос. Выберите макрос, который необходимо назначить кнопке, и щелкните на кнопке ОК.

Чтобы назначить макрос для фигуры, создайте последнюю с помощью инструментов на панели Рисование. Щелкните на фигуре правой кнопкой и выберите команду Назначить макрос из контекстного меню.

Выполнение процедуры по событию

Некоторые процедуры должны выполняться, если в системе (программе) происходит определенное событие. Это может быть открытие рабочей книги, ввод данных в рабочий лист, сохранение книги, щелчок на элементе управления `CommandButton` и многое другое. Процедура, которая выполняется, когда происходит какое-либо событие, называется *обработкой события*. Процедуры обработки событий характеризуются общими свойствами.

- Имеют специальные названия, состоящие из имени объекта, символа подчеркивания и названия события. Например, процедура, которая выполняется при открытии рабочей книги, называется `Workbook_Open`.
- Хранятся в окне кода для определенного объекта.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Процедурам обработки событий посвящена глава 19.

Выполнение процедуры в окне отладки

Процедуру можно также выполнить, введя ее название в окне отладки (Immediate) программы VBE. Если это окно в данный момент не отображено, нажмите <Ctrl+G>. Окно Immediate выполняет операторы VBA, которые вы вводите в его области. Чтобы выполнить процедуру, достаточно ввести название процедуры в окне Immediate и нажать <Enter>.

Этот метод часто применяется при разработке процедуры, поскольку в окне Immediate можно вводить непосредственные команды, чтобы сразу же увидеть результат их выполнения. Продемонстрируем этот прием.

```
Sub ChangeCase()
  MyString = "Это тест"
  MyString = UCase(MyString)
  Debug.Print MyString
End Sub
```

На рис. 9.5 показан результат выполнения процедуры `ChangeCase` в окне Immediate. Оператор `Debug.Print` немедленно отобразит его.

Передача аргументов в процедуры

Аргументы обеспечивают процедуру данными, использующимися в ее инструкциях. Аргумент может *передавать* следующие данные:

- переменная;
- константа;
- массив;
- объект.

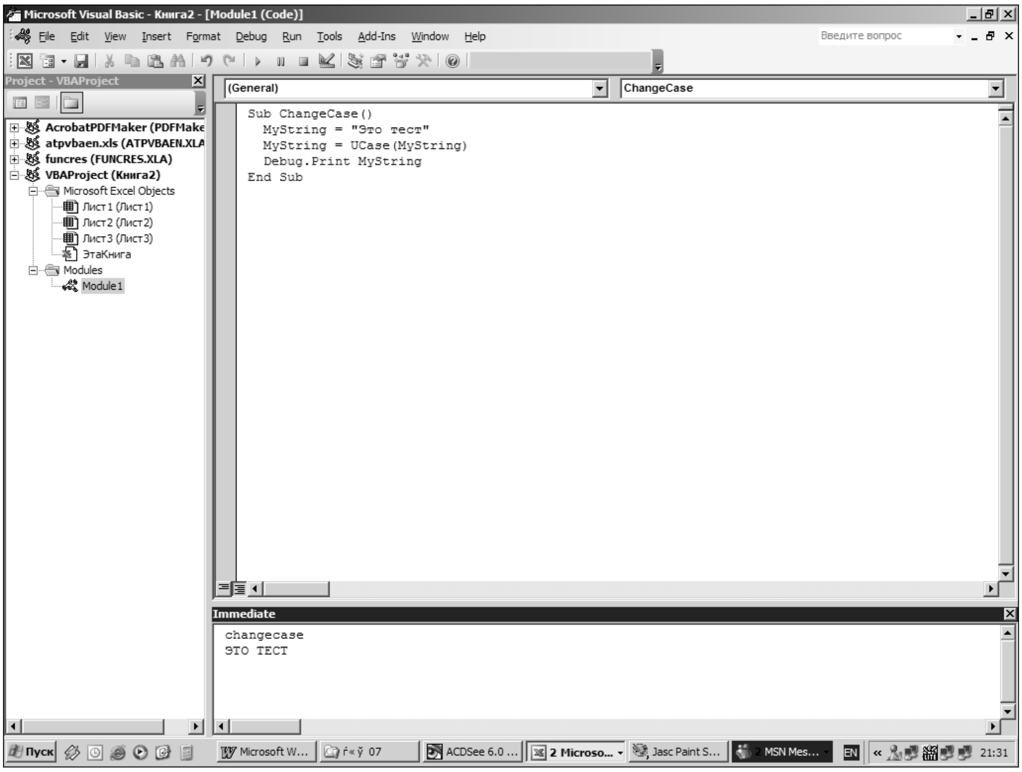


Рис. 9.5. Выполнение процедуры при вводе названия в окне Immediate

Что касается аргументов, процедуры подобны функциям Excel в следующем:

- процедура может не принимать аргументы;
- процедура может иметь фиксированное количество аргументов;
- процедура может иметь неопределенное количество аргументов;
- не все аргументы процедуры являются обязательными;
- все аргументы могут быть необязательными.

Например, существует несколько функций Excel, не имеющих аргументов (например, `СЛЧИС`). Другие функции (такие, как `СЧЕТЕСЛИ`) принимают два аргумента. Функции еще одной группы (например, `СУММ`) могут иметь неограниченное число аргументов (до 30). Некоторые функции Excel имеют необязательные аргументы. Например, функция `ПЛТ` может иметь пять аргументов (три обязательных и два необязательных).

Большинство процедур, с которыми вы сталкивались ранее, объявлялись без аргументов: вводилось ключевое слово `Sub`, после которого указывалось название процедуры и пустые скобки. Пустые скобки обозначают, что процедура не имеет аргументов.

В примере, приведенном далее, отображается две процедуры. Процедура `Main` вызывает процедуру `ProcessFile` трижды (оператор `Call` задается в цикле `For-Next`). Однако перед вызовом `ProcessFile` создается трехэлементный массив. Внутри цикла каждый элемент массива становится аргументом вызываемой процедуры. Процедура

`ProcessFile` принимает один аргумент (с названием `TheFile`). Обратите внимание, что аргумент указывается в скобках в операторе `Sub`. После выполнения `ProcessFile` программа продолжается с оператора, указанного после оператора `Call`.

```
Sub Main
  File(1) = "dept1.xls"
  File(2) = "dept2.xls"
  File(3) = "dept3.xls"
  For i = 1 To 3
    Call ProcessFile(File(i))
  Next i
End Sub
Sub ProcessFile(TheFile)
  Workbooks.Open FileName:=TheFile
'  ...[код]...
End Sub
```

Конечно, вы всегда можете передавать в процедуру текст (не переменные).

```
Sub Main
  Call ProcessFile("budget.xls")
End Sub
```

Существует два способа передачи аргументов в процедуру: по ссылке и по значению. При передаче аргумента по ссылке (этот метод используется по умолчанию) в процедуру передается всего лишь адрес хранения переменной в памяти. При передаче аргумента по значению в процедуру передается *копия* исходной переменной. Следовательно, изменение аргумента в процедуре не вызывает изменения исходной переменной.

Следующий пример подтверждает высказанную концепцию. Аргумент процедуры `Process` передается по ссылке (по умолчанию). После того, как процедура `Main` присваивает переменной `MyValue` значение 10, она вызывает процедуру `Process` и передает `MyValue` в качестве аргумента. Процедура `Process` умножает значение своего аргумента (с названием `YourValue`) на 10. По окончании процедуры `Process` возобновляется выполнение процедуры `Main`, а функция `MsgBox` отображает `MyValue: 100`.

```
Sub Main
  MyValue = 10
  Call Process(MyValue)
  MsgBox MyValue
End Sub

Sub Process(YourValue)
  YourValue = YourValue * 10
End Sub
```

Если необходимо, чтобы вызываемая процедура не изменяла переменные, полученные как аргументы, то измените список аргументов вызываемой процедуры так, чтобы аргументы передавались *по значению*, а не *по ссылке*. Для этого добавьте перед аргументом ключевое слово `ByVal`. Тогда вызываемая процедура будет управлять копией переданных данных, а не самими данными. В следующей процедуре, например, изменения, которые происходят с `YourValue` в процедуре `Process`, не влияют на значение переменной `MyValue` в процедуре `Main`. В результате функция `MsgBox` отображает `10`, а не `100`.

```
Sub Process(ByVal YourValue)
  YourValue = YourValue * 10
End Sub
```

В большинстве случаев вполне подходит метод, используемый по умолчанию, — передача аргументов по ссылке. Однако если процедура призвана изменить данные, которые передаются ей в виде аргументов, а первоначальные данные должны остаться неизмененными, то необходимо использовать передачу данных по значению.

Аргументы процедуры могут быть самыми разными и передаваться и по значению, и по ссылке. Все аргументы, перед которыми указано ключевое слово `ByVal`, передаются по значению; остальные — по ссылке.

НА ЗАМЕТКУ

Если в процедуре применена переменная с пользовательским типом данных, то ее нужно передавать по ссылке. При попытке передать ее по значению происходит ошибка.

Так как в предыдущих примерах не был объявлен тип данных ни для одного аргумента, то все аргументы имеют тип данных `Variant`. Но процедура, использующая аргументы, может определять типы данных непосредственно в списке аргументов. Ниже представлен оператор `Sub` для процедуры с двумя аргументами, имеющими различные типы данных. Первый аргумент объявлен как `Integer`, второй — как `String`.

```
Sub Process(Iterations As Integer, TheFile As String)
```

При передаче аргументов в процедуру важно, чтобы данные, которые передаются в качестве аргументов, имели тип данных аргументов. Например, в предыдущем примере при вызове `Process` и передаче в качестве первого аргумента переменной типа `String` отображается сообщение об ошибке: *ByRef argument type mismatch*.

НА ЗАМЕТКУ

Аргументы имеют как обычные процедуры, так и функции. Однако чаще аргументы используются в функциях. В главе 10, посвященной функциям, приведены дополнительные примеры использования аргументов в процедурах, в том числе рассматривается назначение необязательных аргументов.

Сравнение переменных Public и передачи аргументов в процедуру

В главе 8 отмечалось, что переменная, объявленная как `Public` (вверху модуля), доступна для всех процедур текущего модуля. В некоторых случаях лучше обратиться к переменной `Public`, а не передавать переменную в качестве аргумента при вызове другой процедуры.

Например, приведенная ниже процедура передает значение переменной `MonthVal` в процедуру `ProcessMonth`.

```
Sub MySub()
  Dim MonthVal as Integer
'  ...[код]
  MonthVal = 4
  Call ProcessMonth(MonthVal)
'  ...[код]
End Sub
```

Альтернативный метод будет выглядеть следующим образом:

```
Public MonthVal as Integer

Sub MySub()
'  ...[код]
  MonthVal = 4
  Call ProcessMonth
'  ...[код]
End Sub
```

Во втором фрагменте по причине того, что `MonthVal` является переменной `Public`, процедура `ProcessMonth` имеет к ней доступ, поэтому необходимость в аргументах для этой процедуры отпадает.

Обработка ошибок

Вы, конечно же, догадываетесь, что при выполнении процедуры VBA могут происходить ошибки: синтаксические (которые необходимо исправить перед тем, как выполнять процедуру) и ошибки выполнения (они происходят в процессе выполнения процедуры). В этом разделе рассматривается вторая группа ошибок.

Чтобы процедуры обработки ошибок выполнялись, следует отключить параметр Break on All Errors. В VBE выберите Tools⇨Options, а затем щелкните на вкладке General диалогового окна Options. Если переключатель Break on All Errors установлен, то VBA игнорирует процедуры обработки ошибок. Поэтому обычно используется параметр Break on Unhandled Errors.

Как правило, ошибка в процессе выполнения вызывает остановку программы VBA, а пользователь видит диалоговое окно, в котором отображается номер и описание ошибки. В хорошо написанном приложении вы не столкнетесь с такими служебными сообщениями, так как в приложение включены специальные процедуры, перехватывающие ошибки и выполняющие соответствующие действия. В самом крайнем случае программа обработки ошибок отображает для пользователя более значимое сообщение об ошибке, чем представленное VBA.

Все коды и описание ошибок VBA приведены в приложении В.

Перехват ошибок

Чтобы указать программе, что должно произойти при возникновении ошибки, используется оператор `On Error`. Вы вправе выбрать один из двух вариантов.

- Проигнорировать ошибку и позволить VBA продолжить выполнение программы. После этого можно проанализировать объект `Err`, чтобы узнать, какая ошибка произошла, и при необходимости принять меры по ее предотвращению.
- Перейти к специальному разделу кода для обработки ошибок, чтобы выполнить необходимые действия. Этот раздел вводится в конце процедуры и обозначается специальной меткой.

Чтобы программа продолжала выполняться после возникновения ошибки, необходимо вставить в код следующий оператор.

```
On Error Resume Next
```

Некоторые ошибки несущественны, и их можно запросто игнорировать. Однако сначала следует определить, какая ошибка произошла. При возникновении ошибки можно использовать объект `Err` для определения ее номера. Чтобы отобразить значение `Err.Number` (по умолчанию используется в `Err`), обратитесь к функции `Error`. Например, представленный далее оператор отображает ту же информацию, что и обычное диалоговое окно со сведениями об ошибке Visual Basic (содержит номер ошибки и ее описание).

```
MsgBox "Ошибка" & Err & ": " & Error(Err)
```

На рис. 9.6 показано сообщение об ошибке VBA, а на рис. 9.7 отображена та же ошибка, что и в окне сообщения, но уже в Excel. Конечно, вы можете сделать сообщение об ошибке более значимым для конечных пользователей, используя описательный текст.

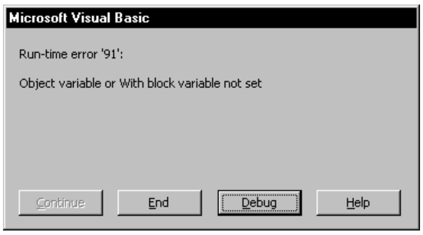


Рис. 9.6. Сообщения об ошибках, используемые в VBA, не всегда дружественны для пользователя

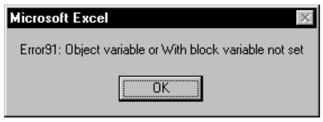


Рис. 9.7. Для отображения кода ошибки и описания можно использовать окно сообщения

Ссылка на `Err` эквивалентна обращению к свойству `Number` объекта `Err`. Следовательно, два приведенных ниже оператора идентичны.

```
MsgBox Err
MsgBox Err.Number
```

Оператор `On Error` также применяется для определения места в процедуре, к которому должна перейти программа в случае ошибки. Чтобы обозначить это место, используется метка.

```
On Error GoTo ErrorHandler
```

Примеры обработки ошибок

В первом примере демонстрируется ошибка, которую можно спокойно проигнорировать. Метод `SpecialCells` выделяет ячейку, которая соответствует заданному критерию. (Действие, эквивалентное выбору команды Правка⇨Перейти и щелчку на кнопке Выделить, чтобы выделить, например, все ячейки, содержащие формулы.)

В приведенном далее примере метод `SpecialCells` выделяет все ячейки в текущем диапазоне, которые содержат формулу, возвращающую число. Если ни одна ячейка в диапазоне не соответствует критерию, VBA генерирует сообщение об ошибке. С помощью оператора `On Error Resume Next` мы предотвращаем появление сообщения об ошибке.

```
Sub SelectFormulas()
  On Error Resume Next
  Selection.SpecialCells(xlFormulas, xlNumbers).Select
  On Error GoTo 0
' ...[код]
End Sub
```

Обратите внимание, что оператор `On Error GoTo 0` восстанавливает нормальную обработку ошибок перед выходом из процедуры.

Следующая процедура использует дополнительный оператор для определения результата: произошла ли ошибка.

```
Sub SelectFormulas2()
  On Error Resume Next
  Selection.SpecialCells(xlFormulas, xlNumbers).Select
  If Err <> 0 Then MsgBox "Не было найдено ячеек с формулами."
  On Error GoTo 0
' ...[код]
End Sub
```

Если значение `Err` не равно нулю, то произошла ошибка, и в диалоговом окне отображается сообщение для пользователя.

Далее продемонстрируем обработку ошибки в результате перехода к метке.

```
Sub ErrorDemo()
  On Error GoTo Handler
  Selection.Value = 123
  Exit Sub
Handler:
  MsgBox "Невозможно присвоить значение выделенному диапазону."
End Sub
```

В процедуре производится попытка присвоить значение текущему выделенному объекту. Если происходит ошибка (например, не выделен диапазон ячеек или лист защищен), то оператор присвоения выдает ошибку. Оператор `On Error` задает переход к метке `Handler` в случае ошибки. Обратите внимание, что перед меткой используется оператор `Exit Sub`. Программа обработки не выполняется, если ошибок не было. Этот оператор можно и не задать — в результате сообщение будет отображаться даже в том случае, если ошибка не происходила.

Иногда ошибка помогает получить определенную информацию. В приведенном ниже примере выполняется проверка того, открыта ли конкретная рабочая книга. При этом обработка ошибок не выполняется.

```
Sub CheckForFile()
  FileName = "BUDGET.XLS"
  FileExists = False

' Цикл по всем рабочим книгам
  For Each book In Workbooks
    If UCase(book.Name) = FileName Then
      FileExists = True
    End If
  Next book

' Отображение соответствующего сообщения
  If FileExists Then _
    MsgBox FileName & " открыт." Else _
      MsgBox FileName & " не открыт."
End Sub
```

В этой процедуре в цикле `For Each-Next` просматриваются все объекты коллекции `Workbooks`. Если книга открыта, переменная `FileExists` получает значение `True`. В результате отображается сообщение, указывающее пользователю, открыта ли рабочая книга.

Предыдущую процедуру можно переписать так, что для определения "открытости" будет использоваться метод обработки ошибок. В следующем примере оператор `On Error Resume Next` указывает VBA игнорировать любые ошибки. В инструкции мы обратимся к рабочей книге, присвоив ей переменную объекта (с помощью ключе-

вого слова `Set`). Если рабочая книга закрыта, происходит ошибка. В структуре `If-Then-Else` проверяется значение свойства `Err` и отображается соответствующее сообщение.

```
Sub CheckForFile()
  Dim x as Workbook
  FileName = "BUDGET.XLS"
  On Error Resume Next
  Set x = Workbooks(FileName)
  If Err = 0 Then
    MsgBox FileNAme & " открыт."
  Else
    MsgBox FileNAme & " не открыт."
  End If
  On Error GoTo 0
End Sub
```

В главе 11 представлены дополнительные примеры кодов, в которых выполняется обработка ошибок.

Реальный пример

В этой главе рассмотрены основные принципы создания процедур. Отметим, что многие приведенные примеры были не очень интересными. Далее представлен пример из реальной жизни. Он иллюстрирует многие концепции, рассмотренные в этой и предыдущих двух главах.

В данном разделе описана среда полезной утилиты, которую можно рассматривать как приложение (согласно определению, предложенному в главе 5). Что более важно, ниже показан *процесс* анализа задачи и последующего ее решения с помощью VBA. Предупреждение опытным пользователям, читающим эту книгу: примите к сведению, что пример рассмотрен в расчете на начинающих. Поэтому мы не только представим код, но и покажем, какими источниками можно пользоваться при разработке программы.

Готовое приложение можно найти на прилагаемом к книге компакт-диске.

Цель

Цель этого упражнения — разработать утилиту, которая изменяет порядок следования листов рабочей книги, сортируя их названия по алфавиту (с помощью одних только функций Excel это сделать невозможно). Если вы часто создаете книги с большим количеством листов, то знаете, что иногда сложно найти интересующий вас лист. Если же их упорядочить по названиям, то любой рабочий лист найти будет значительно проще.

Требования к проекту

С чего начать? Начнем с перечисления требований к приложению. В процессе разработки вы будете обращаться к этому списку для проверки правильности выполнения действий.

Список требований для разрабатываемого приложения приведен далее.

1. Приложение должно сортировать листы (т.е. рабочие листы и листы диаграмм) активной книги по названиям в возрастающем алфавитном порядке.

2. Приложение должно выполняться.
3. Приложение должно быть всегда доступно. Другими словами, пользователь не должен открывать рабочую книгу для использования этой утилиты.
4. Приложение должно правильно выполняться по отношению к любой открытой рабочей книге.
5. В приложении не должны отображаться сообщения об ошибках VBA.

Исходные данные

Часто самой сложной частью проекта является определение того, с чего же начать. В данном случае начнем с перечисления особенностей Excel, которые могут повлиять на выполнение требований к проекту.

- В Excel отсутствует команда сортировки листов. Следовательно, отпадает вариант записи макроса для упорядочивания листов в алфавитном порядке.
- Лист можно легко переместить, перетащив его за ярлык вкладки.

  Примечание: включите функцию записи макросов и перетащите лист в другое место, чтобы узнать, какой код создается при таком действии.
- Следует знать, сколько листов содержится в активной рабочей книге. Эту информацию можно получить с помощью VBA.
- Узнайте названия листов (вновь воспользовавшись VBA).
- В Excel существует команда, сортирующая данные в ячейках рабочего листа.

  Примечание: возможно, стоит перенести названия листов в диапазон ячеек и использовать эту функцию. Или, возможно, в VBA есть метод сортировки, которым можно будет воспользоваться в программе.
- Благодаря диалоговому окну Параметры макроса несложно будет назначить для макроса комбинацию клавиш.
- Если макрос сохранен в личной книге макросов, он всегда доступен.
- Вам понадобится тестировать приложение по мере разработки. Естественно, нельзя тестировать приложение в той же рабочей книге, в которой оно разработано.

  Примечание: создайте фиктивную рабочую книгу, предназначенную специально для тестирования.
- Если разработать программу правильно, то VBA не будет отображать сообщения об ошибках.

  Примечание: не будем принимать желаемое за действительное...

Подход

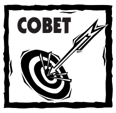
На данном этапе мы точно не знаем, что будем делать дальше, однако можно разработать предварительный, схематический план, описывающий общие задачи.

1. Идентифицировать активную рабочую книгу.
2. Получить список названий всех листов в рабочей книге.
3. Посчитать листы.
4. Отсортировать их (определенным образом).
5. Изменить порядок следования листов в соответствии с параметрами сортировки.

Что необходимо знать

В намеченном плане можно заметить несколько недостатков. Вам необходимо определить, как выполнить следующие операции.

- Идентифицировать активную рабочую книгу.
- Сосчитать листы активной рабочей книги.
- Получить список названий листов.
- Отсортировать список.
- Изменить порядок следования листов в соответствии с отсортированным списком.



Если вам не хватает информации о конкретных методах и свойствах, то обратитесь к материалу этой книги или к электронной справочной системе. Вскоре вы обнаружите то, что вам необходимо. Однако для начала лучше всего включить функцию записи макросов и посмотреть, что записывается в результате выполнения действий, связанных с решением поставленной задачи.

Предварительные результаты записи макросов

Ниже приведен пример, который был получен в результате записи макроса, позволяющей больше узнать о необходимых для решения поставленной задачи средствах VBA. Начнем с рабочей книги, содержащей три рабочих листа. Затем включим функцию записи макросов и укажем, что макрос должен сохраняться в личной книге макросов. В режиме записи перетащим третий рабочий лист на место первого. Результат записи средствами Excel будет следующим.

```
Sub Macro1()
  Sheets("Лист3").Select
  Sheets("Лист3").Move Before:=Sheets(1)
End Sub
```

Найдем в справочной системе слово *Move*: это метод, перемещающий лист в рабочей книге на новое место. Данный метод имеет один аргумент, определяющий будущее положение листа. Таким образом, он имеет непосредственное отношение к нашей задаче.

Вам также необходимо узнать количество листов в активной рабочей книге. Запросим сведения по слову *Count* и определим, что это свойство коллекции. Затем активизируем окно Immediate в VBE и введем такой оператор.

```
? ActiveWorkbook.Count
```

Ошибка! Возможно, требуется посчитать листы в книге.

```
? ActiveWorkbook.Sheets.Count
```

На этот раз успешно. На рис. 9.8 показан результат — еще немного ценной информации.

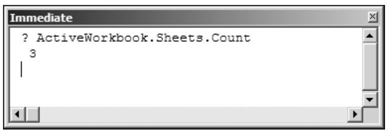


Рис. 9.8. Использование окна Immediate в VBE для тестирования оператора

А что с названиями листов? Проведем еще один тест. Введем в окне Immediate следующий оператор.

```
? ActiveWorkbook.Sheets(1).Name
```

В результате будет получено название первого листа — `Лист3` (так и есть на самом деле). Таким образом, мы получили положительный результат.

Далее, конструкция `For Each-Next` используется для циклического просмотра всех членов коллекции. Просмотрев соответствующий раздел справочной системы, напишем короткую процедуру с использованием этой конструкции.

```
Sub Test()
  For Each Sht In ActiveWorkbook.Sheets
    MsgBox Sht.Name
  Next Sht
End Sub
```

И вновь успешно! Макрос отобразил три окна сообщения, в каждом из которых — новое название листа.

Наконец, пришло время подумать о функциях сортировки. Справочная система укажет, что метод `Sort` относится к объекту `Range`. Поэтому одним из решений задачи будет перенесение названия листов в диапазон ячеек и сортировка этого диапазона. Однако такая задача оказалась слишком сложной для данного приложения. Возможно, целесообразнее сформировать из названий листов массив строк, а затем отсортировать этот массив с использованием кода VBA.

Подготовка

Теперь мы обладаем достаточным количеством информации, для того чтобы приступить к работе над серьезной программой. Однако прежде следует задать первоначальные настройки. Итак, выполним следующие инструкции.

1. Создайте пустую рабочую книгу с пятью рабочими листами: названия Лист1, Лист2, Лист3, Лист4 и Лист5.
2. Разместите листы произвольно, чтобы они следовали не по порядку.
3. Сохраните рабочую книгу как `Test.xls`.
4. Активизируйте VBE и выберите проект `Personal.xls` в окне Project.
5. Если `Personal.xls` не появляется в окне Project, это означает, что вы никогда не использовали личную книгу макросов. Excel создаст для вас эту книгу, когда вы запишете макрос (любой) и определите, что он должен сохраняться в личной книге макросов.
6. Добавьте новый модуль VBA (используя команду Insert⇨Module).
7. Создайте пустую процедуру с названием `SortSheets` (рис. 9.9).
8. Макрос можно сохранить в любом модуле личной книги макросов. Однако лучше хранить каждый макрос в отдельном модуле. Таким образом, вы сможете легко экспортировать модуль и импортировать его в другой проект.
9. Активизируйте Excel. Выполните команду Сервис⇨Макрос⇨Макросы и нажмите кнопку Параметры, чтобы присвоить данному макросу комбинацию клавиш. Хорошим выбором будет <Ctrl+Shift+S>.

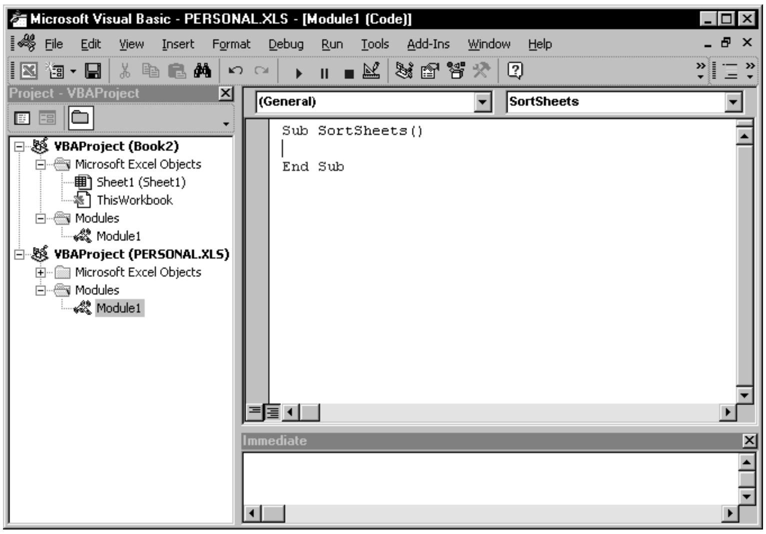


Рис. 9.9. Пустая процедура в модуле, расположенном в Личной книге макросов

Написание кода

Теперь пришло время приступить к написанию программы. Вначале необходимо поместить названия листов в массив строк. Так как пока неизвестно, сколько листов содержит активная рабочая книга, для объявления массива используем оператор `Dim` с пустыми скобками. Помните, что всегда можно применить оператор `ReDim` и изменить размерность массива на требуемое число элементов.

Введем код, вставляющий названия листов в массив `SheetNames`. Кроме того, в цикл добавим функцию `MsgBox`, чтобы убедиться, что названия листов на самом деле вводятся в массив.

```
Sub SortSheets()
    Dim SheetNames() as String
    Dim i as Integer
    Dim SheetCount as Integer
    SheetCount = ActiveWorkbook.Sheets.Count
    ReDim SheetNames(1 To SheetCount)
    For i = 1 To SheetCount
        SheetNames(i) = ActiveWorkbook.Sheets(i).Name
        MsgBox SheetNames(i)
    Next i
End Sub
```

Чтобы проверить эту программу, активизируем рабочую книгу `Test.xls` и нажмем <Ctrl+Shift+S>. Появится пять окон сообщений с названиями листов активной рабочей книги.

Рекомендуем вам тестировать код по мере его создания. Когда вы убедитесь, что программа работает правильно, удалите операторы `MsgBox` (через некоторое время они начнут вас раздражать).

Вместо того чтобы использовать для проверки работы функцию `MsgBox`, можно применить метод `Print` объекта `Debug`, отображающий информацию в окне Immediate. В рассмотренном примере вместо оператора `MsgBox` целесообразнее обратиться к оператору.

```
Debug.Print SheetNames(i)
```

Этот прием может показаться вам не таким навязчивым, как операторы `MsgBox`.

На данном этапе процедура `SortSheets` всего лишь создает массив названий листов в соответствии с порядком указания их в активной рабочей книге. Остается два шага: отсортировать значения в массиве `SheetNames`, а затем изменить порядок следования листов в книге согласно отсортированному массиву.

Создание процедуры сортировки

Переходим к сортировке массива `SheetNames`. Можно вставить программу сортировки в процедуру `SortSheets`, но лучше написать общую процедуру сортировки, которую можно будет потом использовать для управления другими объектами (сортировка массивов — довольно популярная операция).

Возможно, вас несколько обескуражит идея создания процедуры сортировки. Однако не беспокойтесь: несложно найти стандартные процедуры, которые можно непосредственно или с небольшими изменениями использовать в своей программе. Конечно, наиболее полным источником такой информации является Internet.

Существует несколько способов сортировки массивов. Мы выбрали *пузырьковый* метод (хотя это не очень быстрый прием, но его легко запрограммировать). В данном конкретном приложении высокая скорость выполнения операций не так уж важна.

В пузырьковом методе используется вложенный цикл `For-Next`, в котором оценивается каждый элемент массива. Если элемент массива больше, чем следующий, то эти два элемента меняются местами. Такое сравнение повторяется для каждой пары элементов (т.е. $n - 1$ раз).

В главе 11 описаны другие процедуры сортировки и приведено сравнение процедур по скорости выполнения.

Ниже отображена написанная мною процедура сортировки.

```
Sub BubbleSort(List() As String)
'   Сортировка массива List в возрастающем порядке
    Dim First As Integer, Last As Integer
    Dim i As Integer, j As Integer
    Dim Temp As String
    First = LBound(List)
    Last = UBound(List)
    For i = First To Last - 1
        For j = i + 1 To Last
            If List(i) > List(j) Then
                Temp = List(j)
                List(j) = List(i)
                List(i) = Temp
            End If
        Next j
    Next i
End Sub
```

Эта процедура имеет один аргумент: одномерный массив с названием `List`. Массив, который передается в процедуру, может быть любой длины. Для присвоения нижней и верхней границы массива переменным `First` и `Last` использовались функции `LBound` и `UBound`, соответственно.

После того, как мы убедимся, что процедура работает правильно, изменим процедуру `SortSheets`. Для этого добавим вызов процедуры `BubbleSort`, аргументом которой является массив `SheetNames`. На данном этапе модуль будет иметь такой вид.

```
Sub SortSheets()
    Dim SheetNames() As String
    Dim SheetCount as Integer
    Dim i as Integer
    SheetCount = ActiveWorkbook.Sheets.Count
    ReDim SheetNames(1 To SheetCount)
    For i = 1 To SheetCount
        SheetNames(i) = ActiveWorkbook.Sheets(i).Name
    Next i
    Call BubbleSort(SheetNames)
End Sub

Sub BubbleSort(List() As String)
    Dim First As Integer, Last As Integer
    Dim i As Integer, j As Integer
    Dim Temp As String
    First = LBound(List)
    Last = UBound(List)
    For i = First To Last - 1
        For j = i + 1 To Last
            If List(i) > List(j) Then
                Temp = List(j)
                List(j) = List(i)
                List(i) = Temp
            End If
        Next j
    Next i
End Sub
```

По окончании процедуры `SortSheets` образуется массив, состоящий из отсортированных названий листов активной рабочей книги. Чтобы проверить это, можно отобразить содержимое массива в окне Intermediate, добавив в конец процедуры `SortSheets` такой код.

```
For i = 1 To SheetCount
  Debug.Print SheetNames(i)
Next i
```

Пока все идет нормально. Осталось написать программу для изменения порядка следования листов в книге в соответствии с отсортированными элементами массива `SheetNames`.

Программа, записанная раньше, вновь пригодилась. Помните инструкцию, которая была получена при перемещении листа в рабочей книге в первую позицию?

```
Sheets("Лист3").Move Before:=Sheets(1)
```

Далее напишем цикл `For-Next`, который просматривает каждый лист и перемещает его в соответствующее место, указанное в массиве `SheetNames`.

```
For i = 1 To SheetCount
  Sheets(SheetNames(i)).Move Before:=Sheets(i)
Next i
```

Например, в первой итерации цикла счетчик (i) равен 1. Первый элемент массива SheetNames (в данном примере) — лист *Лист1*. Следовательно, выражение для метода Move в цикле будет таким.

```
Sheets("Лист1").Move Sheets(1)
```

Во второй итерации цикла выражение имеет следующий вид.

```
Sheets("Лист2").Move Sheets(2)
```

В процедуру SortSheets добавим новый код.

```
Sub SortSheets()
    SheetCount = ActiveWorkbook.Sheets.Count
    ReDim SheetNames(1 To SheetCount)
    For i = 1 To SheetCount
        SheetNames(i) = ActiveWorkbook.Sheets(i).Name
    Next i
    Call BubbleSort(SheetNames)
    For i = 1 To SheetCount
        ActiveWorkbook.Sheets(SheetNames(i)).Move _
            Before:=ActiveWorkbook.Sheets(i)
    Next i
End Sub
```

Тестирование показало, что процедура прекрасно работает с рабочей книгой Test.xls.

Теперь необходимо собрать весь код. Объявим все переменные, используемые в процедурах, и добавим несколько комментариев, а также пустых строк, чтобы программу можно было легче прочесть. В результате процедура SortSheets будет преобразована к следующему виду.

```
Sub SortSheets()
'  Эта процедура сортирует листы активной рабочей книги
'  в возрастающем порядке.

  Dim SheetNames() as String
  Dim i as Integer
  Dim SheetCount as Integer

'  Определение количества листов, изменение размерности массива
  SheetCount = ActiveWorkbook.Sheets.Count
  ReDim SheetNames(1 To SheetCount)

'  Заполнение массива названиями листов
  For i = 1 To SheetCount
    SheetNames(i) = ActiveWorkbook.Sheets(i).Name
  Next I

'  Сортировка массива в возрастающем порядке
  Call BubbleSort(SheetNames)

'  Перемещение листов
  For i = 1 To SheetCount
    ActiveWorkbook.Sheets(SheetNames(i)).Move _
      ActiveWorkbook.Sheets(i)
  Next i
End Sub
```

Итак, все работает. Чтобы проверить программу дальше, добавим еще несколько листов в книгу Test.xls и изменим некоторые названия. Программа работает прекрасно!

Дополнительное тестирование

Наверное, вы считаете, что работа окончена. Однако тот факт, что процедура работает с рабочей книгой `Test.xls`, не означает, что она будет работать со всеми рабочими книгами. Чтобы проверить программу дальше, загрузим несколько других рабочих книг и вновь запустим программу. Скоро вы убедитесь в том, что приложение неидеально (если быть точным, оно далеко от идеала). Были обнаружены следующие проблемы.

- Рабочие книги с большим количеством листов сортируются очень долго, так как при операциях перемещения окно постоянно обновляется.
- Сортировка не всегда выполняется. Например, в одном из тестов лист с названием *SUMMARY* (все буквы в верхнем регистре) был размещен перед листом под названием *Sheet1*. Эта проблема вызвана процедурой `BubbleSort` (так как *U* в верхнем регистре считается больше, чем *h* в нижнем).
- Если на экране не отображаются окна рабочих книг, при нажатии <Ctrl+Shift+S> макрос выдает ошибку.
- Если структура рабочей книги защищена, метод `Move` не работает.
- Метод `Move` не работает корректно при скрытии рабочих листов. Лист фактически перемещается и размещается перед первым нескрытым листом. Такое неожиданное поведение не задокументировано, поэтому можно сделать вывод, что это дефект Excel.
- После сортировки последний лист рабочей книги становится активным листом. Изменение активного листа — не очень удачное решение проблемы; лучше, если бы активным оставался лист, который был таковым до начала выполнения программы.
- При прерывании макроса с помощью комбинации клавиш <Ctrl+Break> VBA отображает сообщение об ошибке.

Решение проблем

Решить проблему обновления изображения на экране несложно — для этого вставьте в начале процедуры `SortSheets` такую инструкцию.

```
Application.ScreenUpdating = False
```

Можно было бы исправить и проблему с процедурой `BubbleSort`. Для этого используем функцию `UCase`, чтобы преобразовать названия листов в верхний регистр. Таким образом, все сравнения выполняются с названиями в верхнем регистре. Исправленная строка выглядит следующим образом.

```
If UCase(List(i)) > UCase(List(j)) Then
```

СОВЕТ

Иной способ решения “проблемы регистра” — добавить в начале модуля следующий оператор.

```
Option Compare Text
```

В этом случае VBA выполняет сравнение строк на основе не чувствительных к регистру правил сортировки. Другими словами, символ *A* считается тем же, что и *a*.

Чтобы избежать сообщения об ошибке, которое появляется, когда все рабочие книги свернуты, добавим процедуру проверки ошибок. Если не существует активных рабочих книг, происходит ошибка. Применим оператор `On Error Resume Next`, чтобы проигнорировать ошибку, и затем проверим значение `Err`. Если `Err` не равно нулю, это означает, что произошла ошибка. Следовательно, процедура заканчивается. Ниже приведен код проверки ошибок.

```
On Error Resume Next
SheetCount = ActiveWorkbook.Sheets.Count
If Err <> 0 Then Exit Sub ' Нет активной рабочей книги
```

Можно и не использовать оператор `On Error Resume Next`. Следующий оператор непосредственно определяет, свернута ли рабочая книга, и не реализует обработку ошибок.

```
If ActiveWorkBook Is Nothing Then Exit Sub
```

Обычно для защиты структуры рабочей книги имеется серьезная причина. Мы не будем снимать защиту; программа должна отображать предупреждение, чтобы пользователь снял защиту и снова выполнил макрос. Проверку защищенной структуры книги выполнить легко — свойство `ProtectStructure` объекта `WorkBook` возвращает `True`, если книга защищена. Поэтому добавим в проект следующий код.

```
' Проверка защиты структуры рабочей книги
  If ActiveWorkbook.ProtectStructure Then
    MsgBox ActiveWorkbook.Name & " защищена.", _
      vbCritical, "Сортировка листов невозможна."
    Exit Sub
  End If
```

Сложнее решить проблему со скрытыми листами. Возможно такое решение: использовать массив, контролирующий состояние скрытия каждого листа (которое определено свойством `Visible`). Далее можно написать программу отображения листов и снова скрыть листы после окончания сортировки. Определите новый массив (`SheetHidden`) и используйте следующую программу для выявления скрытых листов и их отображения.

```
'  Заполнение массива скрытыми листами
  For i = 1 To SheetCount
    SheetHidden(i) = Not ActiveWorkbook.Sheets(i).Visible
'  Отображение скрытых листов
    If SheetHidden(i) Then ActiveWorkbook.Sheets(i).Visible = True
    Next i
```

Затем, после сортировки, используйте дополнительный код, чтобы вновь скрыть листы, которые были отображены в начале выполнения приложения.

```
'  Повторное скрытие листов
  For i = 1 To SheetCount
    If SheetHidden(i) Then ActiveWorkbook.Sheets(i).Visible = False
  Next i
```

Так как обновление экрана не происходит, то отображение и скрытие листов осуществляется “незаметно”.

Чтобы вновь активизировать рабочий лист после выполнения сортировки, напишите программу, которая присваивает первоначально активный лист переменной объекта (`OldActive`), а затем активизирует лист после окончания программы.

При нажатии <Ctrl+Break> выполнение макроса обычно приостанавливается, и VBA выдает сообщение об ошибке. Но так как одна из целей проекта — избежать сообщений об ошибке, то необходимо вставить команду предотвращения подобной

ситуации. В справочной системе указано, что объект `Application` обладает свойством `EnableCancelKey`, которое может отключить комбинацию клавиш <Ctrl+Break>. Поэтому мы добавим следующий оператор в начало программы.

```
Application.EnableCancelKey = xlDisabled
```

ВНИМАНИЕ

Будьте очень внимательны, когда отключаете прерывание макроса. Если программа попадает в бесконечный цикл, выйти из него вы не сможете. Лучше вставлять этот оператор только после того, как вы убедитесь, что все работает идеально.

После внесения указанных выше исправлений процедура `SortSheets` будет выглядеть так, как показано в листинге 9.1.

Листинг 9.1. Окончательная версия процедуры SortSheets

```
Sub SortSheets()
'   Процедура сортировки листов в активной рабочей книге

    Dim SheetNames() As String
    Dim SheetHidden() As Boolean
    Dim i As Integer
    Dim SheetCount As Integer
    Dim VisibleWins As Integer
    Dim Item As Object
    Dim OldActive As Object

    If ActiveWorkbook Is Nothing Then Exit Sub ' Нет активных книг
    SheetCount = ActiveWorkbook.Sheets.Count

'   Проверка защиты книги
    If ActiveWorkbook.ProtectStructure Then
        MsgBox ActiveWorkbook.Name & " защищена. ", _
           vbCritical, "Невозможно сортировать листы. "
        Exit Sub
    End If

'   Отключение комбинации клавиш Ctrl+Break
    Application.EnableCancelKey = xlDisabled

'   Получение количества листов
    SheetCount = ActiveWorkbook.Sheets.Count

'   Изменение размерности массива
    ReDim SheetNames(1 To SheetCount)
    ReDim SheetHidden(1 To SheetCount)

'   Сохранение ссылки на активный лист
    Set OldActive = ActiveSheet

'   Заполнение массива именами листов
    For i = 1 To SheetCount
        SheetNames(i) = ActiveWorkbook.Sheets(i).Name
    Next i

    For i = 1 To SheetCount
        SheetHidden(i) = Not ActiveWorkbook.Sheets(i).Visible
'       unhide hidden sheets
        If SheetHidden(i) Then ActiveWorkbook.Sheets(i).Visible = True
    Next i

'   Сортировка массива
    Call BubbleSort(SheetNames)
```

```
'   Отключение функции обновления экрана
    Application.ScreenUpdating = False

'   Перемещение листов
    For i = 1 To SheetCount
        ActiveWorkbook.Sheets(SheetNames(i)).Move _
            Before:=ActiveWorkbook.Sheets(i)
    Next i

'   Повторное скрытие листов
    For i = 1 To SheetCount
        If SheetHidden(i) Then ActiveWorkbook.Sheets(i).Visible = False
    Next i

'   Повторная активизация исходного листа
    OldActive.Activate
End Sub
```

Доступность утилиты

Так как макрос `SortSheets` сохранен в личной книге макросов, он всегда доступен при запуске Excel. На этом этапе макрос может выполняться при выборе названия макроса в диалоговом окне Макрос (это окно отображается при нажатии <Alt+F8>) или нажатии <Ctrl+Shift+S>.

Вы также можете назначить макрос для кнопки на панели инструментов или команды меню. Как это сделать, описано ранее в этой главе.

Оценка проекта

Итак, результат получен. Утилита соответствует всем изначальным требованиям: она сортирует все листы в активной рабочей книге, ее можно легко выполнить, она всегда доступна, выполняется (что легко проверить) для всех рабочих книг и пока еще не отображала сообщений об ошибке VBA.

В процедуре все еще присутствует одна небольшая проблема: сортировка достаточно строгая и не всегда кажется “логичной”. Например, после сортировки лист Лист11 размещается перед листом Лист2, хотя большинство пользователей хотят видеть Лист2 перед Лист11.

На компакт-диске, прилагаемом к этой книге, содержится еще одна версия утилиты, в которой решена описанная выше проблема.

Глава 10

Создание функций

В ЭТОЙ ГЛАВЕ...

VBA позволяет создавать два типа структур: процедуры типа `Sub` и процедуры типа `Function`. Процедуры типа `Sub` были описаны в предыдущей главе. В данной — рассматриваются функции.

- Различия между процедурами и функциями
- Создание специальных функций
- О функциях и аргументах функций
- Примеры, примеры, примеры...
- Создание функции, аналогичной функции `СУММ` в Excel
- Отладка функций, работа с диалоговым окном Мастер функции, использование надстроек для хранения пользовательских функций
- Вызов функций Windows API для выполнения сложных действий

ПЕРЕКРЕСТНАЯ ССЫЛКА

В главе 11 представлены полезные и часто используемые примеры функций. Вы можете воспользоваться многими из них в своей работе.

Процедуры и функции: сравнение

Процедуру можно рассматривать как команду, которая выполняется пользователем или другой процедурой. С другой стороны, процедуры типа `Function` обычно возвращают отдельное значение (или массив), подобно функциям рабочих листов Excel и встроенным функциям VBA. Как и встроенные функции, процедуры типа `Function` имеют аргументы.

Процедуры типа `Function` универсальны и используются в двух ситуациях.

- Как часть выражения в процедуре VBA.
- В формулах, которые создаются на рабочем листе.

Процедуру типа `Function` можно использовать везде, где применяются функции Excel и встроенные функции VBA.

Назначение пользовательских функций

Несомненно, вам знакомы функции Excel. Даже начинающие пользователи знают, как работать с самыми популярными функциями в формулах рабочего листа — `СУММ`, `СРЗНАЧ` и `ЕСЛИ`. Excel содержит более 300 встроенных функций, а также дополнительные функции, доступные после установки надстройки Пакет анализа. Если их недостаточно, то можете создать специальные функции с помощью VBA.

Зная о существовании большого количества функций, доступных в Excel и VBA, вы можете заняться и созданием новых функций. Некоторые из них смогут упростить жизнь конечным пользователям. Тщательно спланированные специальные функции эффективно используются в формулах на рабочем листе и в процедурах VBA.

Например, зачастую пользовательские функции создаются, чтобы сократить формулы. Короткие формулы намного легче воспринимаются и обрабатываются. Однако следует отметить, что специальные функции, используемые в формулах, обычно выполняются медленнее, чем встроенные.

При создании приложений обратите внимание на то, что в некоторых процедурах часто повторяются одни и те же вычисления. В таком случае стоит подумать о создании специальной функции, выполняющей эти вычисления, и впоследствии ее достаточно будет лишь вызвать из процедуры. Специальная функция, например, поможет заменить повторяющийся код программы и уменьшить возможные ошибки.

Созданные вами функции окажутся полезными и для ваших коллег. Другие программисты, возможно, захотят приобрести специальные функции, которые экономят рабочее время и затрачиваемые на их программирование усилия.

Вас может пугать мысль о создании пользовательских функций, однако этот процесс несложен. Например, я *люблю* создавать пользовательские функции. Мне особенно нравится, если созданные мною функции отображаются в диалоговом окне Мастер функций наряду со встроенными функциями Excel. Появляется ощущение, будто вы самостоятельно обновляете программное обеспечение.

В этой главе вы узнаете об этапах создания пользовательских функций; причем теория подкреплена полезными примерами.

Простой пример функции

Чтобы сразу же перейти к делу, рассмотрим пример функции VBA.

Пользовательская функция

Ниже приведена пользовательская функция, определенная в модуле VBA. Эта функция имеет название `Reverse` и принимает один аргумент. Функция переставляет символы аргумента в обратном порядке и возвращает результат как строковое значение.

```
Function Reverse(InString) As String
' Возвращает символы аргумента в обратном порядке
 Dim i as Integer, StringLength as Integer
 Reverse = ""
 StringLength = Len(InString)
 For i = StringLength To 1 Step -1
  Reverse = Reverse & Mid(InString, i, 1)
 Next i
End Function
```

ВНИМАНИЕ При создании пользовательских функций, которые используются в формуле рабочего листа, убедитесь, что код вводится в обычном модуле VBA. Если вы поместите пользовательские функции в модуле листа `Лист` или модуле `ЭтаКнига`, они не будут выполняться в формулах.

Использование функции на рабочем листе

При вводе формулы, в которой используется функция `Reverse`, Excel выполняет программу для получения конечного значения. Эту функцию можно использовать в следующей формуле.

```
=Reverse(A1)
```

Примеры данной функции в действии показаны на рис. 10.1. Формулы введены в столбце B, в качестве аргумента используется текст в столбце A. Функция возвращает свой аргумент с представленными в обратном порядке символами.



Рис. 10.1. Использование пользовательской функции в формуле рабочего листа

Фактически, такая функция действует подобно любой встроенной функции рабочего листа. Вы можете вставить функцию в формулу, используя команду Вставка⇨Функция или кнопку Вставка функции. В диалоговом окне Мастер функций пользовательские функции по умолчанию расположены в категории Определенные пользователем.

Вы также можете создавать вложенные пользовательские функции и сочетать их в формулах с другими элементами. Например, приведенная ниже формула (абсолютно бесполезная) использует функцию `Reverse` дважды. В результате будет получена первоначальная строка.

```
=Reverse(Reverse(A1))
```

Использование функции в процедуре VBA

Следующая процедура VBA, определенная в том же модуле, что и пользовательская функция `Reverse`, сначала отображает окно для ввода текста пользователем. Затем процедура использует встроенную функцию VBA `MsgBox` для отображения данных, введенных пользователем, но уже после их обработки функцией `Reverse` (рис. 10.2). Первоначальные данные отображаются в заголовке окна сообщения.

```
Sub ReverseIt()
 Dim UserInput as String
 UserInput = InputBox("Введите текст:")
 MsgBox Reverse(UserInput), , UserInput
End Sub
```

В примере, показанном на рис. 10.2, в ответ на функцию `InputBox` была введена строка `Профессиональное программирование на VBA в Excel`. Функция `MsgBox` отображает текст-перевертыш.

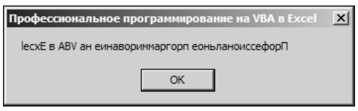


Рис. 10.2. Применение пользовательской функции в процедуре VBA

Анализ пользовательской функции

Функции могут быть достаточно сложными, если это необходимо. Как правило, они более сложны и намного более полезны, чем процедура в приведенном ниже примере. Тем не менее, анализ этого примера поможет вам разобраться в данном вопросе.

Приведем текст функции.

```
Function Reverse(InString) As String
' Возвращает символы аргумента в обратном порядке
 Dim i as Integer, StringLength as Integer
 Reverse = ""
 StringLength = Len(InString)
 For i = StringLength To 1 Step -1
  Reverse = Reverse & Mid(InString, i, 1)
 Next i
End Function
```

Обратите внимание, что функция начинается с ключевого слова `Function`, а не `Sub`, после которого указывается название функции (`Reverse`). Эта специальная функция использует только один аргумент (`InString`), заключенный в скобки. `As String` определяет тип данных значения, которое возвращает функция. (Excel по умолчанию использует тип данных `Variant`, если тип данных не определен.)

Вторая строка — простой комментарий (необязательный), который описывает выполняемые функцией действия. После комментария приведен оператор `Dim` с двумя переменными (`i` и `StringLength`), применяемыми в функции.

Затем функция инициализирует результат как пустую строку. Обратите внимание, что название функции используется как переменная. Когда функция заканчивает выполнение, она всегда возвращает текущее значение переменной, соответствующей названию функции.

Далее функция VBA `Len` определяет длину введенной строки и присваивает значение переменной `StringLength`.

Следующие три инструкции составляют цикл `For-Next`. Функция циклически и в обратном порядке просматривает каждый символ введенного текста и создает на их основе новую строку. Обратите внимание, что значение шага `Step` в цикле `For-Next` — отрицательное число, поэтому итерации цикла выполняются в обратном порядке. Инструкция цикла использует функцию VBA `Mid`, которая возвращает один символ из строки, введенной пользователем. По окончании цикла переменная `Reverse` будет хранить все символы введенной строки, переставленные в обратном порядке. Эта строка — значение, которое возвращает функция.

Функция заканчивается оператором `End Function`.

Задачи, которые не выполняются с помощью пользовательских функций рабочего листа

При разработке пользовательских функций важно понимать различие между функциями, которые вызываются из других процедур VBA, и функциями, используемыми в формулах рабочего листа. Функции, используемые в формулах рабочего листа, — "пассивные". Например, такая функция не может обрабатывать диапазоны или изменять содержимое рабочего листа. Вы детально разберетесь в этом вопросе, решая практические задачи.

Вы можете попробовать написать пользовательскую функцию рабочего листа, изменяющую формат ячейки. Например, рекомендуется всегда иметь формулу, использующую специальную функцию для изменения цвета текста в зависимости от значения в ячейке. Но, как бы вы ни пытались, такую функцию написать невозможно. Что бы вы ни делали, функция всегда будет возвращать ошибку. Помните: функция возвращает значение, но не может выполнять действия над объектами.

Синтаксис функции

Пользовательские функции имеют много общего с процедурами типа `Sub`. (Детально о процедурах типа `Sub` рассказано в главе 9.)

Объявление функции

Для объявления функции используется следующий синтаксис.

```
[Public | Private][Static] Function имя ([список_аргументов])[As тип]
 [инструкции]
 [имя = выражение]
 [Exit Function]
 [операторы]
 [имя = выражение]
End Function
```

В этом синтаксисе используются следующие ключевые слова.

- `Public` (необязательное ключевое слово) — указывает, что функция доступна для других процедур во всех остальных модулях активных проектов VBA.
- `Private` (необязательное ключевое слово) — указывает, что функция доступна только для других процедур в текущем модуле.
- `Static` (необязательное ключевое слово) — указывает, что значения переменных, которые объявлены в функции, сохраняются между вызовами функции.
- `Function` (обязательное ключевое слово) — обозначает начало функции, возвращающей значение или другие данные.
- *`имя`* (обязательное ключевое слово) — представляет произвольное название функции. При именовании функций используются те же правила, что и при задании имен переменным. По окончании выполнения функции результат присваивается ее названию.
- *`список_аргументов`* (необязательный) — список одной или более переменных, представляющих аргументы, которые передаются в функцию. Аргументы заключены в скобки. Для разделения пар аргументов используется запятая.
- *`тип`* (необязательный) — тип данных, который возвращает функция.
- *`операторы`* (необязательные) — корректные инструкции VBA.
- `Exit Function` (необязательный) — оператор, вызывающий немедленный выход из функции до ее завершения.
- `End Function` (обязательное) — ключевое слово, обозначающее конец функции.

О пользовательских функциях, написанных на VBA, необходимо знать следующее: значение всегда присваивается названию функции минимум один раз и, как правило, тогда, когда функция завершила выполнение.

Создание пользовательской функции начните с создания модуля VBA (можно также использовать существующий модуль). Введите ключевое слово `Function`, после которого укажите название функции и список ее аргументов (если они есть) в скобках. Вы также можете объявить тип данных значения, которое возвращает функция, используя ключевое слово `As` (это не обязательно, но рекомендуется). Вставьте код VBA, выполняющий требуемые действия, и проверьте, что необходимое значение присваи-

вается переменной процедуры соответствующей названию функции, минимум один раз в теле функции. Функция заканчивается оператором `End Function`.

Имена функций подчиняются тем же правилам, что и имена переменных. Если вы планируете использовать функцию в формуле рабочего листа, убедитесь, что название не имеет форму адреса ячейки (например, в формуле не будет выполняться функция с названием `J21`). Кроме того, избегайте имен функций, которые совпадают с названиями встроенных функций Excel. В случае конфликта имен Excel всегда будет отдавать предпочтение встроенной функции.

Область действия функции

В главе 9 была рассмотрена концепция области действия процедуры (`Public` или `Private`). Это же относится и к функциям: область действия функции определяет, может ли она быть вызвана процедурами в других модулях или рабочих листах.

Ниже представлены условия, о которых следует помнить при определении области действия функции.

- Если область действия функции не задана, то по умолчанию подразумевается `Public`.
- Функции, объявленные как `Private`, не отображаются в диалоговом окне Мастер функций. Следовательно, при создании функции, которая должна использоваться только в процедуре VBA, необходимо объявить ее как `Private`, чтобы пользователи не пытались применять ее в формуле.
- Если в программе VBA необходимо вызвать функцию, которая определена в другой рабочей книге, задайте ссылку на другую рабочую книгу, воспользовавшись командой VBE Tools⇨References.

Выполнение функций

Процедуру можно выполнить несколькими способами, однако функции выполняются только одним из двух вариантов.

- В результате вызова из другой процедуры.
- При использовании в формуле рабочего листа.

ВЫЗОВ ИЗ ПРОЦЕДУРЫ

Пользовательские функции можно вызывать из процедуры точно так же, как и встроенные функции. Например, после определения функции с названием `SumArray` в код вводится оператор, показанный ниже.

```
Total = SumArray(MyArray)
```

Этот оператор выполняет функцию `SumArray` с аргументом `MyArray`, возвращает результат функции и присваивает его переменной `Total`.

Кроме того, можно использовать метод `Run` объекта `Application`.

```
Total = Application.Run ("SumArray", "MyArray")
```

Первый аргумент метода `Run` — имя функции. Дополнительные аргументы представляют аргумент (аргументы) функции. Аргументами метода `Run` могут быть строки (как показано выше), числа или переменные.

ИСПОЛЬЗОВАНИЕ В ФОРМУЛЕ РАБОЧЕГО ЛИСТА

Применение специальных функций в формуле рабочего листа во многом подобно применению встроенных функций, за некоторым исключением. Удостоверитесь, что Excel может найти необходимую функцию. Если функция сохранена в той же рабочей книге, то ничего особенного делать не нужно. Если же функция находится в другой рабочей книге, то необходимо указать Excel, где ее найти.

Вы вправе выбрать один из трех способов.

- *Указать перед названием функции ссылку на файл.* Например, если вы решили использовать функцию с названием `CountNames`, определенную в открытой рабочей книге `Myfunc.xls`, то обратитесь к следующей ссылке.

  ```
  =Myfunc.xls!CountNames(A1:A1000)
  ```

- Если вы вставите функцию с помощью команды Вставка⇨Функция, то ссылка на рабочую книгу будет добавлена автоматически.
- *Установить ссылку на рабочую книгу.* Это можно сделать с помощью команды VBE Tools⇨References. Если функция определена в ссылаемой рабочей книге, то не следует указывать имя рабочего листа. Даже если зависимая рабочая книга определена по ссылке, то в диалоговом окне Мастер функции указывается рабочая книга, содержащая функцию (хотя это не обязательно).
- *Создать надстройку.* При создании надстройки на основе рабочей книги, в которой представлены и функци, можно не задавать ссылку на файл, если одна из функций используется в формуле. Однако надстройку необходимо устанавливать. Детально надстройки рассматриваются в главе 21.

Изобретаем колесо

Большинство встроенных функций Excel в VBA создать невозможно. Однако *некоторые* функции можно дублировать.

Развлечения ради я написал собственную версию функции Excel `ПРОПИСН` (преобразовывающую символы строки в верхний регистр) и назвал ее `UpCase`. (К сожалению, она правильно применяется только к английским словам и выражениям. – *Прим. ред.*)

```
Function UpCase(InString As String) As String
' Преобразует символы аргумента в верхний регистр
 Dim StringLength As Integer
 Dim i As Integer
 Dim ASCIIVal As Integer
 Dim CharVal As Integer

 StringLength = Len(InString)
 UpCase = InString
 For i = 1 To StringLength
  ASCIIVal = Asc(Mid(InString, i, 1))
  CharVal = 0
  If ASCIIVal >= 97 And ASCIIVal <= 122 Then
   CharVal = —32
   Mid(UpCase, i, 1) = Chr(ASCIIVal + CharVal)
  End If
 Next i
End Function
```

Итак, мне было интересно, чем пользовательская функция отличается от встроенной функции, поэтому я создал рабочий лист, вызывающий функцию 10000 раз с аргументом длиной в 26 символов. Вычисления продолжались 13 секунд. Затем пользовательская функция была заменена на встроенную функцию Excel `ПРОПИСН`. Вновь проведем тест и убедимся, что вычисления выполнены меньше чем за секунду.

Функция `UpCase` не является оптимальным алгоритмом для выполнения поставленной задачи, однако можно с уверенностью сказать, что пользовательская функция никогда не будет работать так же быстро, как встроенные функции VBA.

Вы заметите, что, в отличие от процедур, функции не отображаются в диалоговом окне Макрос при выполнении команды Сервис⇨Макрос⇨Макросы. Кроме того, функцию нельзя выбрать при использовании команды VBE Run⇨Sub/UserForm (или нажатии <F5>), если курсор установлен в тексте функции (так как при этом отображается диалоговое окно Макрос, где вы можете выбрать макрос для выполнения). В результате вам необходимо выполнить дополнительную работу по тестированию функций еще в процессе разработки. Один из возможных подходов — создать простую процедуру, вызывающую функцию. Если функция должна использоваться в формулах рабочего листа, то для ее проверки следует ввести простую формулу.

Аргументы функций

Всегда помните о следующих особенностях аргументов процедур функций.

- Аргументы могут представляться переменными (в том числе и массивами), константами, символьными данными или выражениями.
- Некоторые функции не имеют аргументов.
- Определенные функции имеют строго заданное число обязательных аргументов (от 1 до 60).
- Отдельные функции имеют как обязательные, так и необязательные аргументы.

Если формула содержит пользовательскую функцию рабочего листа и возвращает `#ЗНАЧ!`, то в функции есть ошибка. Ошибка может быть вызвана логическими ошибками в программе, передачей в функцию некорректных аргументов или выполнением ошибочного действия (например, попытка изменить формат ячейки). См. раздел “Отладка функций” далее в этой главе.

Примеры функций

В настоящем разделе приведен ряд примеров, иллюстрирующих эффективное использование аргументов функций. Данный материал касается также и процедур.

Все примеры функций из этого раздела можно найти на прилагаемом к книге компакт-диске.

Функция без аргументов

Как и процедуры, пользовательские функции не обязательно должны иметь аргументы. Например, в Excel есть несколько встроенных функций, не использующих аргументы, в том числе функции `СЛЧИС()` и `СЕГОДНЯ()`. Несложно создать аналогичные функции.

Ниже приведен пример простой функции, не использующей аргументов. Следующая функция возвращает свойство `UserName` объекта `Application`. Это имя отображается в диалоговом окне Excel Параметры вкладки Общие и хранится в системном реестре Windows.

```
Function User()
' Возвращает имя зарегистрированного пользователя
 User = Application.UserName
End Function
```

При вводе следующей формулы ячейка возвращает имя текущего пользователя (предполагается, что оно нормально сохранено в реестре).

```
=User()
```

При использовании функции без аргументов в формуле рабочего листа необходимо поставить пустые скобки. Это требование не обязательно, если вы вызываете функцию в процедуре VBA. Однако включение пустых скобок делает понятным, что вызывается функция.

Чтобы использовать данную функцию в другой процедуре, ее следует присвоить переменной, применить в выражении или использовать как аргумент другой функции.

В следующем примере вызывается функция `User` и в качестве аргумента оператора `MsgBox` используется значение, которое она возвращает. Оператор конкатенации (`&`) объединяет текстовую строку с результатом функции `User`.

```
Sub ShowUser()
 MsgBox "Ваше имя — " & User()
End Sub
```

Еще одна функция без аргументов

Функция Excel `СЛЧИС()` служит для быстрого заполнения диапазона ячеек значениями. Однако при этом случайные числа изменяются при каждом пересчете формул на рабочем листе. Поэтому обычно приходится преобразовывать формулы в значения с помощью команды Правка⇨Специальная вставка с активным параметром Значения.

Вы можете создать пользовательскую функцию, возвращающую случайные числа, которые не будут изменяться при пересчете формул. Для этого используйте встроенную функцию VBA `Rnd()`, которая возвращает случайное число в диапазоне от 0 до 1. Эта функция будет выглядеть следующим образом.

```
Function StaticRand()
' Возвращает случайное число, которое
' не изменяется при пересчете формул
 StaticRand = Rnd()
End Function
```

Если вы будете генерировать последовательность случайных целых чисел в диапазоне от 0 до 1000, то обратитесь к формуле

```
=ЦЕЛОЕ(StaticRand()*1000)
```

Значения, полученные с ее помощью, никогда не изменяются, в отличие от сгенерированных встроенной функцией `СЛЧИС()`.

Управление пересчетом функций

Вы, наверное, хотели бы узнать, когда при использовании пользовательской функции в формуле рабочего листа пересчитывается ее значение.

Пользовательские функции ведут себя подобно встроенным функциям Excel. Обычно специальная функция пересчитывается, только когда это необходимо — т.е. в случае изменения одного из аргументов функции. Однако вы можете выполнять пересчет функций чаще. Функция пересчитывается при изменении любой ячейки, если в процедуру добавлен следующий оператор.

```
Application.Volatile True
```

Метод `Volatile` объекта `Application` имеет один аргумент (`True` или `False`). Если функция отмечена как `Volatile` (т.е. изменяемая), она пересчитывается всякий раз, когда изменяется ячейка рабочего листа.

Например, поведение пользовательской функции `NonStaticRand` можно изменить с помощью метода `Volatile` так, чтобы оно напоминало поведение функции Excel `СЛЧИС()`.

```
Function NonStaticRand()
' Возвращает случайное число, которое
' изменяется при каждом пересчете
 Application.Volatile True
 NonStaticRand = Rnd()
End Function
```

При использовании аргумента `False` метода `Volatile` функция пересчитывается только тогда, когда в результате пересчета изменяется один из ее аргументов (если функция не имеет аргументов, этот метод не выполняется).

Чтобы вызвать полный пересчет формул, в том числе и неизменных пользовательских функций, нажмите <Ctrl+Alt+F9>. Эта комбинация клавиш, к примеру, генерирует новые случайные числа с помощью функции `StaticRand`, представленной выше в этой главе.

Функция с одним аргументом

В этом разделе описана функция, используемая для подсчета комиссионных с объема продаж одного из менеджеров. Вычисления основываются на следующей таблице значений.

Продажи за месяц	Ставка комиссионных
0–$9999	8,0%
$10000–$19999	10,5%
$20000–$39999	12,0%
$40000 и больше	14,0%

Обратите внимание, что ставка комиссионных изменяется нелинейно и зависит от общего объема продаж за месяц. Служащие, которые продают больше, получают более высокие комиссионные.

Существует несколько способов вычислить комиссионные для различных объемов продаж, введенных в таблицу Excel. Потратив немного времени, вы можете получить длинную формулу, подобную приведенной ниже.

```
=ЕСЛИ(И(A1>=0;A1<=9999,99);A1*0,08;
 ЕСЛИ(И(A1>=10000;A1<=19999,99);A1*0,105;
 ЕСЛИ(И(A1>=20000;A1<=39999,99);A1*0,12;
 ЕСЛИF(A1>=40000;A1*0,14;0))))
```

Такая идея неудачна по нескольким причинам. Во-первых, формула чрезмерно сложна, и ее нелегко понять. Во-вторых, значения строго определены в формуле, из-за чего ее сложно изменять.

Рекомендуется применить описанный ниже подход (не требующий программирования на VBA): использовать для вычисления ставки комиссионных функцию просмотра таблицы соответствия `ВПР`.

```
=ВПР(A1;таблица;2)*A1
```

Еще лучше (тогда не нужно использовать таблицу соответствия) создать пользовательскую функцию, представленную ниже.

```
Function Commission(Sales)
 Const Tier1 = 0.08
 Const Tier2 = 0.105
 Const Tier3 = 0.12
 Const Tier4 = 0.14
' Вычисляет комиссионные с объема продаж
 Select Case Sales
  Case 0 To 9999.99: Commission = Sales * Tier1
  Case 10000 To 19999.99: Commission = Sales * Tier2
  Case 20000 To 39999.99: Commission = Sales * Tier3
  Case Is >= 40000: Commission = Sales * Tier4
 End Select
End Function
```

После ввода этой функции в модуль VBA ее можно использовать в формуле на рабочем листе или вызвать из других процедур VBA.

При вводе в ячейку следующей формулы будет получен результат 3000 (объем продаж 25000 соответствует ставке комиссионных 12%).

```
=Commission(25000)
```

Даже если на рабочем листе не требуется применять пользовательские функции, их создание значительно упрощает программирование на VBA. Например, если процедура VBA вычисляет комиссионные с продаж, то можно использовать точно такую же функцию и вызвать ее из процедуры VBA. Ниже приведена небольшая процедура, запрашивающая у пользователя объем продаж и использующая функцию `Commission` для вычисления ставки комиссионных.

```
Sub CalcComm()
 Dim Sales as Long
 Sales = InputBox("Введите объем продаж:")
 MsgBox "Комиссионные составляют " & Commission(Sales)
End Sub
```

Процедура `CalcComm` сначала отображает окно ввода данных и запрашивает у пользователя объем продаж. Затем она отображает окно сообщения с вычисленными комиссионными для введенного объема продаж.

Данная процедура выполняется, но она далека от совершенства. Ниже показана усовершенствованная версия процедуры, отображающая отформатированные значения и предлагающая повторить цикл вычислений, пока пользователь не щелкнет на кнопке Нет (рис. 10.3).

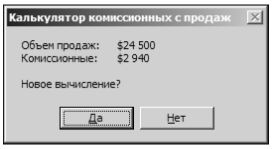


Рис. 10.3. Использование функции для отображения результата вычислений

```
Sub CalcComm()
 Dim Sales as Long
 Dim Msg As String, Ans As String

' Запрос ввести объем продаж
 Sales = Val(InputBox("Введите объем продаж:", _
  "Калькулятор комиссионных с продаж"))

' Создание сообщения
 Msg = "Объем продаж: " & vbTab & Format (Sales, "$#,##0.00")
 Msg = Msg & vbCrLf & "Комиссионные:" & vbTab
 Msg = Msg & Format(Commission(Sales), "$#,##0.00")
 Msg = Msg & vbCrLf & vbCrLf & "Новое вычисление?"

' Отображение результата и запрос следующего вычисления
 Ans = MsgBox(Msg, vbYesNo, "Калькулятор комиссионных с продаж")
 If Ans = vbYes Then CalcComm
End Sub
```

Эта функция использует две встроенные константы VBA: `vbTab` обозначает табуляцию (чтобы отделить результат), а `vbCrLf` определяет возврат каретки (для перехода на следующую строку). Функция VBA `Format` указывает значение в заданном формате (в данном случае со знаком доллара, пробелом для разделения разрядов и двумя десятичными знаками).

В обоих этих примерах функция `Commission` должна содержаться в активной рабочей книге; в противном случае Excel отобразит сообщение об ошибке, указывающее, что функция не определена.

Функция с двумя аргументами

Представим, что менеджер, о котором речь шла выше, отображает новую политику, разработанную для уменьшения текучести кадров: общая сумма комиссионных, подлежащих выплате, увеличивается на 1% за каждый год, который служащий проработал в компании.

Изменим специальную функцию `Commission` (описанную в предыдущем разделе) так, чтобы она принимала два аргумента. Новый аргумент представляет количество лет, отработанных сотрудником в компании. Назовем эту новую функцию `Commission2`.

```
Function Commission2(Sales, Years)
' Вычисляет комиссионные с продаж
' на основе рабочего стажа в компании
 Const Tier1 = 0.08
 Const Tier2 = 0.105
 Const Tier3 = 0.12
 Const Tier4 = 0.14
 Select Case Sales
  Case 0 To 9999.99: Commission2 = Sales * Tier1
  Case 10000 To 19999.99: Commission2 = Sales * Tier2
  Case 20000 To 39999.99: Commission2 = Sales * Tier3
  Case Is >= 40000: Commission2 = Sales * Tier4
 End Select
 Commission2 = Commission2 + (Commission2 * Years / 100)
End Function
```

Довольно просто, правда? Добавим второй аргумент (`Years`) в оператор `Function` и применим дополнительные вычисления, изменяющие формулу комиссионных.

Далее приведен пример написания формулы с использованием этой функции (предполагается, что объем продаж задан в ячейке `A1`, а количество лет, которые проработал в компании рассматриваемый служащий, указано в ячейке `B1`).

```
=Commission2(A1;B1)
```

Функция с аргументом в виде массива

Функции могут в качестве аргументов принимать также один или несколько массивов, обрабатывать этот массив (массивы) и возвращать единственное значение. Функция, представленная ниже, принимает как аргумент массив и возвращает сумму его элементов.

```
Function SumArray(List) As Double
 Dim Item As Variant
 SumArray = 0
 For Each Item In List
  If WorksheetFunction.IsNumber(Item) Then _
   SumArray = SumArray + Item
 Next Item
End Function
```

Функция Excel `IsNumber` проверяет, является ли каждый элемент числом, прежде чем добавить его к общему целому. Добавление этого простого оператора проверки данных устраняет ошибки несоответствия типов при попытке выполнить арифметическую операцию над строкой.

Следующая процедура демонстрирует, как вызвать эту функцию в процедуре. Процедура `MakeList` создает 100-элементный массив и присваивает каждому элементу массива случайное число. Функция `MsgBox` отображает сумму значений массива в результате вызова функции `SumArray`.

```
Sub MakeList()
 Dim Nums(1 To 100) As Double
 Dim i as Integer
 For i = 1 To 100
  Nums(i) = Rnd * 1000
 Next i
 MsgBox SumArray(Nums)
End Sub
```

Так как функция `SumArray` не объявляет тип данных своего аргумента (и аргумент имеет тип `Variant`), то она работает также и в формулах рабочего листа. Например, следующая формула возвращает сумму значений в диапазоне `A1:C10`.

```
=SumArray(A1:C10)
```

Возможно, вы заметили, что при использовании в формуле функция `SumArray` подобна функции Excel `СУММ`. Единственное отличие заключается в том, что функция `SumArray` принимает только один аргумент (а в `СУММ` может использоваться до 30-ти аргументов). Не стоит забывать, что этот пример приведен только в целях обучения. Функция `SumArray` в формуле не имеет абсолютно никаких преимуществ перед функцией Excel `СУММ`.

Функция с необязательными аргументами

Многие встроенные функции Excel имеют необязательные аргументы. Пример — функция `ЛЕВСИМВ`, возвращающая символы с левого края строки. Она имеет следующий синтаксис.

ЛЕВСИМВ(*текст*[;*кол_символов*])

Первый аргумент — обязательный, а второй — нет. Если не указан второй аргумент, Excel предполагает значение 1. Следовательно, две формулы, приведенные ниже возвращают одинаковые результаты.

```
ЛЕВСИМВ(A1;1)
ЛЕВСИМВ(A1)
```

Пользовательские функции, разработанные в VBA, также могут иметь необязательные аргументы. Необязательный аргумент вы зададите, если введете перед именем аргумента ключевое слово `Optional`. В списке аргументов необязательные аргументы определяются после всех обязательных.

Далее приведен пример пользовательской функции с необязательным аргументом.

```
Function User(Optional UpperCase As Variant)
     If IsMissing(UpperCase) Then UpperCase = False
     If UpperCase =True Then
          User = Ucase(Application.UserName)
     Else
          User = Application.UserName
     End If
End Function
```

Если аргумент равен `False` или опущен, то имя пользователя возвращается без каких-либо изменений. Если же аргумент функции `True`, то имя пользователя возвращается в символах верхнего регистра (с помощью VBA-функции `Ucase`). Обратите внимание на первый оператор функции — он содержит VBA-функцию `IsMissing`, которая определяет наличие аргумента.

Приведенные ниже формулы справедливы, а первые две возвращают одинаковый результат.

```
=User()
=User(False)
=User(True)
```

НА ЗАМЕТКУ

Если вам необходимо определить, подставлялся ли необязательный аргумент в функцию, то объявите этот аргумент с типом данных `Variant`. Только в этом случае можно использовать встроенную функцию `IsMissing`, как показано в предыдущем примере.

Далее приведен пример еще одной пользовательской функции с необязательным аргументом. Эта функция случайным образом выбирает одну ячейку из диапазона данных и возвращает содержимое этой ячейки. Если второй аргумент имеет значение `True`, то выделенное значение изменяется при каждом пересчете рабочего листа. Если второй аргумент имеет значение `False` (или не задан), функция не пересчитывается, кроме тех случаев, когда изменяется одна из ячеек диапазона введенных данных.

```
Function Draw(Rng As Variant, Optional Recalc As Boolean = False)
' Случайным образом выбирает одну ячейку из диапазона

' Функция изменяемая, если Recalc имеет значение True
  Application.Volatile Recalc

' Определить случайную ячейку
  Draw = Rng(Int((Rng.Count) * Rnd + 1))
End Function
```

Обратите внимание, что второй аргумент функции `Draw` включает ключевое слово `Optional`, а также значение по умолчанию.

Все приведенные ниже формулы корректны, причем первые две возвращают одинаковые результаты.

```
=Draw(A1:A100)
=Draw(A1:A100;False)
=Draw(A1:A100;True)
```

Эта функция может быть полезной для выбора лотерейных номеров, победителя из списка имен и т.д.

Функция VBA, возвращающая массив

VBA содержит весьма полезную функцию с названием `Array`. Она возвращает значение с типом данных `Variant`, которое содержит массив (т.е. несколько значений). Если вы знакомы с формулами массивов в Excel, то сможете легко разобраться в функции `Array` в VBA. Формула массива вводится в ячейку после нажатия <Ctrl+Shift+Enter>. Excel добавляет вокруг формулы скобки, чтобы указать, что это формула массива. Более подробную информацию о формулах массивов вы найдете в главе 3.

Важно понимать, что массив, возвращаемый функцией `Array`, — это не тот обычный массив, который составлен из элементов с типом данных `Variant`. Другими словами, массив `Variant` — не то же самое, что массив значений типа `Variant`.

Функция `MonthNames`, приведенная ниже, — простой пример применения функции `Array` в пользовательской функции.

```
Function MonthNames()
 MonthNames = Array("Январь", "Февраль", "Март", "Апрель", _
  "Май", "Июнь", "Июль", "Август", "Сентябрь", "Октябрь", _
  "Ноябрь", "Декабрь")
End Function
```

Функция `MonthNames` возвращает горизонтальный массив названий месяцев. Вы можете создать формулу массива для нескольких ячеек, используя функцию `MonthNames`. Прежде чем ее использовать, убедитесь, что в модуле VBA введен код функции. Затем на рабочем листе выделите несколько ячеек в строке (для начала выделите 12 ячеек), введите следующую формулу и нажмите <Ctrl+Shift+Enter>.

```
=MonthNames()
```

Если необходимо сгенерировать вертикальный массив названий месяцев, выделите вертикальный диапазон, введите следующую формулу и нажмите <Ctrl+Shift+Enter>.

```
=ТРАНСП(MonthNames())
```

Данная формула использует функцию Excel `ТРАНСП` для преобразования горизонтального массива в вертикальный.

Следующий пример — вариация на тему функции `MonthNames`.

```
Function MonthNames(Optional MIndex)
 Dim AllNames As Variant
 AllNames = Array("Январь", "Февраль", "Март", "Апрель", _
  "Май", "Июнь", "Июль", "Август", "Сентябрь", "Октябрь", _
  "Ноябрь", "Декабрь")
 If IsMissing(MIndex) Then
  MonthNames = AllNames
 Else
  Select Case MIndex
   Case Is >= 1
'    Определить значение месяца (например, 13=1)
    MonthVal = ((MIndex - 1) Mod 12)
    MonthNames = AllNames(MonthVal)
   Case Is <= 0 ' Вертикальный массив
    MonthNames = Application.Transpose(AllNames)
  End Select
 End If
End Function
```

Обратите внимание, что для проверки незаданного аргумента используется функция VBA `IsMissing`. В данной ситуации невозможно задать значение по умолчанию для незаданного аргумента в списке аргументов функции, так как значение по умолчанию определяется в функции. Функцию `IsMissing` можно использовать, только если необязательный аргумент имеет тип `Variant`.

Эта усовершенствованная функция использует необязательный аргумент, который необходим для выполнения некоторых задач.

- Если аргумент не задан, функция возвращает горизонтальный массив названий месяцев.
- Если аргумент меньше или равен 0, функция возвращает вертикальный массив названий месяцев. Для преобразования массива используется функция Excel `ТРАНСП` (`Transpose`).
- Если аргумент больше или равен 1, функция возвращает название месяца, соответствующее значению аргумента.



В этой процедуре использован уже немного знакомый вам оператор `Mod` для определения значения месяца. Оператор `Mod` возвращает остаток от деления первого операнда на второй. Например, аргумент 13 возвращает 0, аргумент 24 возвращает 11 и т.д.

Вы можете использовать данную функцию разными способами, как показано на рис. 10.4.

Диапазон `A1:L1` содержит следующую формулу, введенную как массив. Для начала выделите `A1:L1`, затем введите формулу и нажмите <Ctrl+Shift+Enter>.

```
=MonthNames()
```

В диапазоне `A3:A14` находятся целые числа от 1 до 12. Ячейка `B3` содержит обычную формулу, которая скопирована в 11 ячеек, следующих за ней.

```
=MonthNames(A3)
```

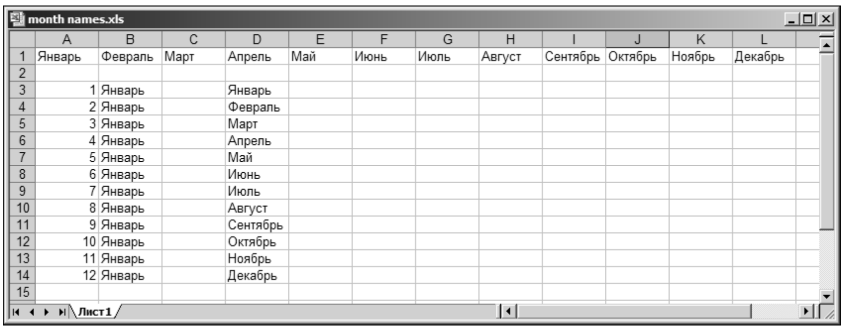
В диапазоне `D3:D14` располагается формула, введенная как массив.

```
=MonthNames(-1)
```

Помните, что для ввода формулы массива необходимо нажать <Ctrl+Shift+Enter>.



Нижняя граница массива, созданная с помощью функции `Array`, определяется нижней границей, заданной в операторе `Option Base` вверху модуля. Если оператор `Option Base` не задан, то по умолчанию используется нижняя граница 0.



month names.xls

	A	B	C	D	E	F	G	H	I	J	K	L
1	Январь	Февраль	Март	Апрель	Май	Июнь	Июль	Август	Сентябрь	Октябрь	Ноябрь	Декабрь
2												
3	1	Январь		Январь								
4	2	Январь		Февраль								
5	3	Январь		Март								
6	4	Январь		Апрель								
7	5	Январь		Май								
8	6	Январь		Июнь								
9	7	Январь		Июль								
10	8	Январь		Август								
11	9	Январь		Сентябрь								
12	10	Январь		Октябрь								
13	11	Январь		Ноябрь								
14	12	Январь		Декабрь								
15												

Лист1

Рис. 10.4. Несколько способов передачи в рабочий лист массива или одного значения

Функция, возвращающая значение ошибки

В некоторых случаях необходимо, чтобы пользовательская функция возвращала значение ошибки. Рассмотрим функцию `Reverse`, которая была представлена ранее в этой главе.

```
Function Reverse(InString) As String
' Возвращает обратную строку
 Dim i as Integer, StringLength as Integer
 Reverse = ""
 StringLength = Len(InString)
 For i = StringLength To 1 Step -1
  Reverse = Reverse & Mid(InString, i, 1)
 Next i
End Function
```

При использовании в формуле на рабочем листе эта функция переставляет символы своего аргумента (текста или значения) в обратном порядке. Предположим, вы хотите, чтобы функция работала только с текстовыми строками. Если аргумент содержит нестроковый тип данных, то необходимо, чтобы функция возвращала значение ошибки (`#Н/Д`).

Возможно, потребуется создать строку, которая выглядит как значение ошибки в формуле Excel.

```
Reverse = "#Н/Д"
```

Несмотря на то, что строка *выглядит* как значение ошибки, она не обрабатывается как ошибка другими формулами, которые могут на нее ссылаться. Чтобы получить в результате выполнения функции *настоящее* значение ошибки, используйте функцию `CVErr`, которая преобразует номер ошибки в настоящую ошибку.

К счастью, в VBA содержатся встроенные константы для обозначения ошибок, которые должна возвращать пользовательская функция. Эти значения — ошибки выполнения формул Excel, а не ошибки выполнения кода VBA. Ниже приведен список встроенных констант ошибок.

- `xlErrDiv0` (для ошибки `#ДЕЛ/0!`).
- `xlErrNA` (для ошибки `#Н/Д`).
- `xlErrName` (для ошибки `#ИМЯ?`).
- `xlErrNull` (для ошибки `#ПУСТО!`).
- `xlErrNum` (для ошибки `#ЧИСЛО!`).
- `xlErrRef` (для ошибки `#ССЫЛ!`).
- `xlErrValue` (для ошибки `#ЗНАЧ!`).

Чтобы получить ошибку `#Н/Д` в пользовательской функции, примените следующий оператор.

```
Reverse = CVErr(xlErrNA)
```

Ниже приведена переделанная функция `Reverse`. Она вызывает функцию Excel `ЕТЕКСТ (IsText)` для определения, содержит ли аргумент текст. Если ячейка содержит текст, то функция возвращает нормальный результат. Если же ячейка содержит не текст (или пуста), то функция возвращает ошибку `#Н/Д`.

```
Function Reverse(InString) as Variant
' Если аргумент - строка, возвращает обратную строку
' В противном случае возвращает ошибку #N/A
 Dim i as Integer, StringLength as Integer
 If Application.WorksheetFunction.IsText(InString) Then
  Reverse = ""
  StringLength = Len(InString)
  For i = StringLength To 1 Step -1
   Reverse = Reverse & Mid(InString, i, 1)
  Next i
 Else
  Reverse = CVErr(xlErrNA)
 End If
End Function
```



Обратите внимание, что был также изменен тип данных для значения, которое возвращает функция. Так как функция может теперь возвращать что-то еще, кроме строки, то я изменил тип данных на `Variant`.

Функция с неопределенным количеством аргументов

Некоторые функции Excel имеют неопределенное количество аргументов. Знакомый пример — функция `СУММ`, которая имеет следующий синтаксис.

```
СУММ(число1;число2;..)
```

Первый аргумент обязателен, но можно задать до 29-ти дополнительных аргументов. Ниже приведен пример функции `СУММ` с четырьмя аргументами-диапазонами.

```
СУММ(A1:A5;C1:C5;E1:E5;G1:G5)
```

Допускается использовать в функции разные типы аргументов. Например, следующий пример иллюстрирует использование трех аргументов: первый — диапазон, второй — значение, а третий — выражение.

```
=СУММ(A1:A5;12;24*3)
```

Существует возможность создавать функции, имеющие неопределенное количество аргументов. Основная идея заключается в следующем: примените в качестве последнего (или единственного) массив и добавьте перед ним ключевое слово `ParamArray`.



`ParamArray` относится только к *последнему* аргументу в списке аргументов процедуры. Он всегда определяет тип данных `Variant` и только необязательный аргумент (хотя ключевое слово `Optional` не используется).

Ниже представлена функция, которая может иметь произвольное количество одномерных аргументов (она не поддерживает многомерные аргументы-диапазоны). Функция возвращает сумму аргументов.

```
Function SimpleSum(ParamArray arglist() As Variant) As Double
 For Each arg In arglist
  SimpleSum = SimpleSum + arg
 Next arg
End Function
```

Функция `SimpleSum` далеко не такая гибкая, как функция Excel `СУММ`. Протестируйте ее на практике с разными типами аргументов, и вы убедитесь, что функция выдает ошибку, если все аргументы не являются либо числовым значением, либо ссылкой на ячейки, содержащие числовые значения.

Создание аналога функции Excel СУММ

В данном разделе представлена пользовательская функция с названием MySum. В отличие от функции SimpleSum, рассмотренной в предыдущем разделе, функция MySum идеально копирует поведение функции Excel СУММ.

Перед рассмотрением кода функции MySum уделим внимание функции Excel СУММ. Данная функция имеет очень широкие возможности. Она насчитывает до 30-ти аргументов (даже пропущенные аргументы), причем аргументами могут выступать числовые значения, ячейки, диапазоны, представленные в виде текста числа, логические значения и даже встроенные функции. Рассмотрим следующую формулу.

```
=СУММ(B1;5;"6";;ИСТИНА;КОРЕНЬ(4);A1:A5)
```

Данная полностью корректная формула содержит все типы аргументов, перечисленные в порядке их использования в функции:

- ссылку на одну ячейку;
- символьное значение;
- строку, которая выглядит как числовое значение;
- пропущенный аргумент;
- логическое значение ИСТИНА;
- выражение, использующее другую функцию;
- ссылку на диапазон.

Функция MySum (листинг 10.1) также обрабатывает все эти типы аргументов.

Рабочую книгу, которая содержит функцию MySum, можно найти на прилагаемом к книге компакт-диске.

Листинг 10.1. Функция MySum

```
Function MySum(ParamArray args() As Variant) As Variant
' Эмуляция функции Excel СУММ

' Объявление переменных
  Dim i As Variant
  Dim TempRange As Range, cell As Range
  Dim ECode As String
  MySum = 0

' Обработка каждого аргумента
  For i = 0 To UBound(args)
'   Пропуск аргументов
    If Not IsMissing(args(i)) Then
'       Определение типа аргумента
        Select Case TypeName(args(i))
            Case "Range"
'               Создание временного диапазона для строки/столбца
                Set TempRange = Intersect(args(i).Parent.UsedRange, args(i))
                For Each cell In TempRange
                    If IsError(cell) Then
                        MySum = cell ' return the error
                        Exit Function
                    End If
                    If cell = True Or cell = False Then
```

```
                    MySum = MySum + 0
                Else
                    If IsNumeric(cell) Or IsDate(cell) _
                       Then MySum = MySum + cell
                End If
            Next cell
        Case "Null"  'игнорирование
        Case "Error" 'возвращение к ошибке
            MySum = args(i)
            Exit Function
        Case "Boolean"
            If args(i) = "True" Then MySum = MySum + 1
        Case "Date"
            MySum = MySum + args(i)
        Case Else
             MySum = MySum + args(i)
    End Select
  End If
 Next i
End Function
```

При анализе кода функции MySum помните о следующем.

- Пропущенные аргументы (определенные с помощью функции IsMissing) просто игнорируются.
- Процедура использует для определения типа аргумента (Range, Error и т.д.) функцию VBA TypeName. Каждый тип аргумента обрабатывается определенным способом.
- Для аргумента-диапазона функция циклически просматривает все ячейки диапазона и прибавляет их значения к сумме.
- Функция имеет тип данных Variant, так как она должна возвращать ошибку, если один из ее аргументов имеет значение ошибки.
- Если аргумент содержит ошибку (например, #ДЕЛ!/0), то функция MySum возвращает ошибку — как и функция Excel СУММ.
- Функция Excel СУММ считает, что текстовая строка имеет значение 0, если только она не является символьным аргументом (т.е. фактическим значением, а не переменной). Следовательно, функция MySum прибавляет значение ячейки лишь в том случае, если его можно оценить как число (для этого применяется функция VBA IsNumeric).
- Для аргументов-диапазонов функция использует метод Intersect с целью создания временного диапазона, который состоит из пересечения заданного диапазона и используемого диапазона листа. В результате обрабатываются те случаи, когда аргумент состоит из полной строки или столбца, а его просмотр будет длиться вечность.

Возможно, вас интересует относительная скорость выполнения функций СУММ и MySum. Конечно, функция MySum значительно медленнее, ее скорость зависит от быстродействия вашей системы и самих формул. Если в системе рабочий лист с 1000-ей формул СУММ вычисляется за долю секунды, то после замены функций СУММ на функции MySum пересчет занимает 12 секунд. Функция MySum несколько улучшена, но ее производительность все же намного меньше, чем у функции СУММ.

Надеемся, вы понимаете, что цель этого примера — не создать новую функцию СУММ. Скорее, этот пример показывает, как создавать пользовательские функции рабочих листов, которые выглядят и работают так же, как встроенные функции Excel.

Отладка функций

При использовании формулы на рабочем листе для тестирования процедуры функции происходящие в процессе выполнения ошибки не отображаются в знакомом диалоговом окне сообщений. Формула просто возвратит значение ошибки (`#ЗНАЧ!`). К счастью, это не представляет большой проблемы при отладке функций, так как всегда существует несколько "обходных путей".

- *Поместить в стратегически важных местах функцию `MsgBox`, чтобы контролировать значение отдельных переменных.* Удобно, если в процессе выполнения функций окна сообщений все-таки появляются, в отличие от окон ошибок. Убедитесь, что ваша функция используется только в одной формуле на рабочем листе, иначе окна сообщений будут появляться для каждой такой формулы (подобное поведение быстро надоедает).
- *Протестировать функцию, вызвав ее из процедуры, а не в формуле рабочего листа.* Ошибки в процессе выполнения отображаются обычным образом, поэтому можно либо сразу решить проблему (если она вам известна), либо перейти к отладке.
- *Определить точку остановки в функции и просмотреть функцию пошагово.* При этом можно воспользоваться всеми стандартными инструментами отладки. Чтобы добавить точку остановки, поместите курсор в операторе, в котором вы решили приостановить выполнение, и выберите команду Debug⇨Toggle Breakpoint (или нажмите <F9>).
- *Использовать в программе один или несколько временных операторов `Debug.Print`, чтобы отобразить значения в окне Immediate редактора VBA.* Например, чтобы проконтролировать циклически изменяемое значение, используйте следующий метод.

```
Function VowelCount(r)
 Count = 0
 For i = 1 To Len(r)
  Ch = UCase(Mid(r, i, 1))
  If Ch Like "[AEIOU]" Then
   Count = Count + 1
   Debug.Print Ch, i
  End If
 Next i
 VowelCount = Count
End Function
```

В данном случае значения двух переменных, `Ch` и `i`, выводятся в окне Immediate всякий раз, когда в программе встречается оператор `Debug.Print`. На рис. 10.5 показан результат для случая, когда функция принимает аргумент `Mississippi`.



Рис. 10.5. Использование окна Immediate для отображения результатов во время работы функции

Работа с диалоговым окном Мастер функции

Диалоговое окно Excel Мастер функции — очень удобный инструмент. При создании формулы рабочего листа он позволяет выбрать необходимую функцию из списка (рис. 10.6). Функции в списке группируются в разные категории, таким образом облегчая их поиск. Кроме того, в диалоговом окне Мастер функции отображаются пользовательские функции и предоставляется справка по аргументам функций.

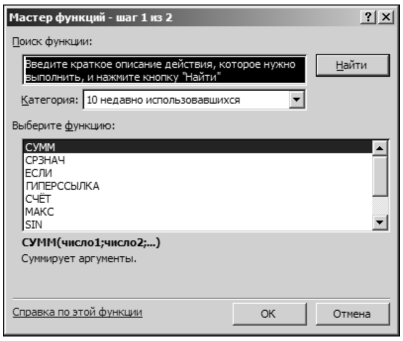


Рис. 10.6. Вставка пользовательской функции в формулу

Пользовательские функции, определенные с помощью ключевого слова `Private`, не отображаются в диалоговом окне Мастер функции (хотя их можно вводить в формулы вручную). Если вы разрабатываете функцию исключительно для использования в других процедурах VBA, то необходимо объявить ее, применив ключевое слово `Private`.

По умолчанию пользовательские функции помещены в категорию Определенные пользователем, но при желании можно перенести их в другую категорию. Кроме того, вы вправе добавить текст, описывающий функцию (рекомендуем это сделать).

В версиях Excel, предшествующих Excel 2002, диалоговое окно Мастер функций называлось Вставка функции. Это диалоговое окно в Excel 2002 было усовершенствовано, но выглядит оно по-другому и имеет команду поиска функции по ключевому слову. К сожалению, команду поиска нельзя применять для поиска пользовательских функций, созданных на VBA.

Определение категории функции

Как ни странно, но Excel не позволяет напрямую определить пользовательскую функцию в одну из категорий. Если вы хотите, чтобы пользовательская функция находилась в категории, отличной от Определенные пользователем, то вам придется прибегнуть к программированию на VBA.

Следующий оператор добавляет функцию с названием `Commission` в категорию Финансовые — первую категорию списка.

```
Application.MacroOptions Macro:="Commission", Category:=1
```

Указанный оператор необходимо выполнить только один раз (а не каждый раз, когда открывается рабочая книга). С этого момента при каждом открытии рабочей книги функция будет помещаться в определенной вами категории.

В табл. 10.1 перечислены номера категорий, которые можно использовать в коде VBA. Обратите внимание, что некоторые из этих категорий (с 10 по 13), как правило, не отображаются в диалоговом окне Мастер функции. Если вы добавите пользовательскую функцию в одну из них, то она будет отображена в диалоговом окне.

Таблица 10.1. Категории функций

Номер категории	Название категории
0	Полный алфавитный перечень
1	Финансовые
2	Дата и время
3	Математические
4	Статистические
5	Ссылки и массивы
6	Работа с базой данных
7	Текстовые
8	Логические
9	Информационные
10	Команды
11	Настройка
12	Управление макросами
13	Динамический обмен данными/Внешние
14	Определенные пользователем
15	Инженерные

Добавление описания функции

При выборе функции в диалоговом окне Мастер функции появляется краткое описание этой функции (рис. 10.7). Вы можете задать описание для пользовательской функции двумя способами: использовать диалоговое окно Макрос или написать программу VBA.

Если вы не задали описание пользовательской функции, то в диалоговом окне Мастер функции будет отображен текст Справка недоступна. Конечно, в большинстве случаев это разочаровывает.

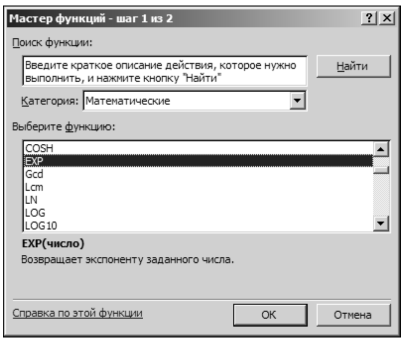


Рис. 10.7. В диалоговом окне Excel Мастер функции отображается краткое описание функций

ОПИСАНИЕ ФУНКЦИИ В ДИАЛОГОВОМ ОКНЕ МАКРОС

Выполните следующие действия, чтобы добавить описание пользовательской функции.

1. Создайте функцию в VBE.
2. Активизируйте Excel, выберите команду Сервис⇨Макрос⇨Макросы.
3. В диалоговом окне Макрос перечислены доступные процедуры, но функции не отображаются в списке.
4. Введите название функции в поле Имя макроса.
5. Щелкните на кнопке Параметры, чтобы отобразить диалоговое окно Параметры макроса.
6. Введите описание функции в специальное поле, как показано на рис. 10.8. Поле Сочетание клавиш к описанию функции не относится.

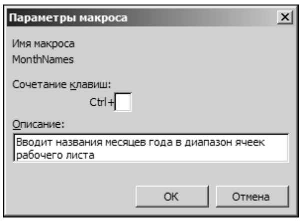


Рис. 10.8. Введение описания функции в диалоговом окне Параметры макроса

7. Щелкните на кнопке ОК, затем — на кнопке Отмена.

После выполнения указанных действий в диалоговом окне Мастер функции при выделении данной функции будет отображено описание, которое вы ввели в п. 5.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Информацию о создании специального раздела справочной системы, доступного в окне Мастер функций, вы найдете в главе 24.

Если для ввода функции используется диалоговое окно Мастер функций, то диалоговое окно Аргументы функции отображается после щелчка на кнопке ОК. Для встроенных функций диалоговое окно Аргументы функции предоставляет описание каждого аргумента. К сожалению, отобразить такое описание для аргументов пользовательских функций невозможно.

ОПИСАНИЕ ФУНКЦИИ С ПОМОЩЬЮ VBA

Второй способ добавить описание специальной функции — написать программу VBA. Представленный ниже оператор назначает описание функции `Commission`.

```
Application.MacroOptions _
  Macro:= "Commission", _
  Description:= "Вычисляет комиссионные с продаж"
```

Этот оператор необходимо выполнить только один раз (а не каждый раз, когда открывается рабочая книга).

Использование надстроек для хранения пользовательских функций

Вы можете по желанию сохранить часто используемые пользовательские функции в файле надстройки. Основное преимущество такого подхода заключается в следующем: функции могут быть применены в формулах без спецификатора имени файла.

Предположим у вас есть пользовательская функция с названием `ZapSpaces`, она хранится в файле `Myfuncs.xls`. Чтобы применить ее в формуле другой рабочей книги (отличной от `Myfuncs.xls`), необходимо ввести следующую формулу.

```
=Myfuncs.xls!ZapSpaces(A1:C12)
```

Если вы создадите надстройку на основе файла `Myfuncs.xls` и эта надстройка будет загружена в текущем сеансе работы Excel, то ссылку на файл можно пропустить, введя следующую формулу.

```
=ZapSpaces(A1:C12)
```



О надстройках речь пойдет в главе 21.

Использование функций Windows API

VBA может заимствовать методы из других файлов, которые не имеют ничего общего с Excel или VBA — например, файлы DLL (Dynamic Link Library — динамически присоединяемая библиотека), которые используются Windows и другими программами. В результате в VBA появляется возможность выполнять операции, которые без заимствованных методов находятся за пределами возможностей языка.

Windows API (Application Programming Interface — прикладной программный интерфейс) представляет собой набор функций, доступных программистам в среде Windows. При вызове функции Windows из VBA вы обращаетесь к Windows API. Многие ресурсы Windows, используемые программистами Windows, можно получить из файлов DLL, в которых хранятся программы и функции, подсоединяемые в процессе выполнения программы, а не во время компиляции.

В Excel, например, тоже есть несколько DLL. Код многих из них можно было добавить непосредственно в исполняемый файл программы — `excel.exe`, но разработчики решили сохранить его в формате DLL, чтобы загружать только при необходимости. Такой подход делает главный исполняемый файл Excel меньше. Кроме того, более эффективно используется память, так как библиотека загружается только при необходимости.

Файлы DLL также применяются для совместного использования кода. Например, во многих программах Windows имеются диалоговые окна для открытия и сохранения файлов. В Windows есть DLL, в которой хранится код создания нескольких стандартных диалоговых окон. Программисты, таким образом, могут вызвать необходимую DLL, а не писать собственные процедуры.

Если вы программируете на языке C, то умеете создавать собственные DLL и использовать их в VBA. В языке Visual Basic от Microsoft тоже поддерживается возможность создания файлов DLL, которые можно вызывать из Excel.

Примеры использования Windows API

Прежде чем использовать функцию Windows API, ее необходимо объявить вверху программного модуля. Если программный модуль — это не стандартный модуль VBA (т.е. модуль для `UserForm`, `Лист` или `ЭтаКнига`), то функцию API необходимо объявить как `Private`.

Объявление функции API имеет некоторую сложность; она должна объявляться максимально точно. Оператор объявления указывает VBA следующее.

- Какую функцию API вы используете.
- В какой библиотеке расположена функция API.
- Аргументы функции API.

После объявления функцию API можно использовать в программе VBA.

Определение папки Windows

Ниже показан пример объявления функции API.

```
Declare Function GetWindowsDirectoryA Lib "kernel32" _
 (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Эта функция, имеющая два аргумента, возвращает название папки, в которой установлена операционная система Windows (с помощью только VBA такое действие выполнить невозможно). После вызова этой функции путь к папке Windows будет храниться в переменной `lpBuffer`, а длина строки пути — в переменной `nSize`.

После вставки оператора `Declare` вверху модуля вы можете обратиться к функции `GetWindowsDirectoryA`. Ниже следует пример вызова функции и отображения результата в окне сообщения.

```
Sub ShowWindowsDir()
  Dim WinPath As String
  Dim WinDir As String
  WinPath = Space(255)
  WinDir = Left(WinPath, GetWindowsDirectoryA _
   (WinPath, Len(WinPath)))
  MsgBox WinDir, vbInformation, "Папка Windows"
End Sub
```

В процессе выполнения процедуры `ShowWindowsDir` отображается окно сообщения с указанием расположения папки Windows. Обычно Windows устанавливается в папке `C:\Windows`, но это не обязательно. Windows NT часто, но не всегда, устанавливается в папке `C:WINNT`.

Иногда требуется создать *оболочку* (*wrapper*) для функций API. Другими словами, вы создадите собственную функцию, использующую функцию API. Такой подход существенно упрощает использование функции API. Ниже приведен пример такой функции VBA.

```
Function WindowsDir() As String
'  Возвращает путь к папке Windows
  Dim WinPath As String
  WinPath = Space(255)
  WindowsDir = Left(WinPath, GetWindowsDirectoryA _
   (WinPath, Len(WinPath)))
End Function
```

После объявления эту функцию можно будет вызвать из другой процедуры.

```
MsgBox WindowsDir
```

Данная функция также используется в формуле рабочего листа.

```
=WindowsDir()
```

Элементы API выполняют действия, которые другим способом выполнить невозможно (или это очень сложно). Если приложение должно найти путь к папке Windows, вы можете целый день искать и не найти в Excel или VBA функцию, которая выполняет эту задачу. Но, зная, как получить доступ к функциям Windows API, вы без особого труда решите свою проблему.

Не удивляйтесь сбоям в системе при использовании в VBA функций Windows API. Заранее сохраните свою работу перед тестированием.

Определение состояния клавиши <Shift>

Приведем еще один пример. Предположим, вы написали макрос VBA, который будет выполняться с помощью кнопки на панели инструментов. Необходимо, чтобы этот макрос выполнялся по-другому, если пользователь при щелчке на кнопке удерживает клавишу <Shift>. Чтобы узнать о нажатии клавиши <Shift>, можно использовать функцию API `GetKeyState`. Функция `GetKeyState` сообщает о том, нажата ли конкретная клавиша. Функция имеет один аргумент, `nVirtKey`, который представляет код интересующей вас клавиши.

В главе 11 вы найдете дополнительные примеры использования API-функций.

Ниже приведена программа, которая выявляет, что при выполнении процедуры обработки события `Button_Click` была нажата клавиша <Shift>. Обратите внимание, что для определения состояния клавиши <Shift> используется константа (принимающая шестнадцатеричное значение), которая затем применяется как аргумент функции `GetKeyState`. Если `GetKeyState` возвращает значение меньше 0, это означает, что клавиша <Shift> нажата; в противном случае клавиша <Shift> не нажата.

```
Declare Function GetKeyState Lib "user32"
 (ByVal nVirtKey As Long) As Integer

Sub Button_Click()
  Const VK_SHIFT As Integer = &H10
  If GetKeyState(VK_SHIFT) < 0 Then
    MsgBox "Shift нажата"
  Else
    MsgBox "Shift не нажата"
  End If
End Sub
```

Рабочая книга на прилагаемом к книге компакт-диске демонстрирует, как определить следующие клавиши (и любые их комбинации): <Ctrl>, <Shift>, <Alt>.

Дополнительная информация о функциях Windows API

Работа с функциями Windows API может быть довольно сложной. Во многих книгах по программированию перечислены операторы объявления функций API с соответствующими примерами. Как правило, можно просто скопировать выражения объявления и использовать функции, не вникая в их суть. Большинство VBA-

программистов в Excel рассматривают функции API как панацею для решения большинства задач. В Internet вы найдете сотни вполне надежных примеров, которые можно скопировать и вставить в собственную программу.

На компакт-диске, прилагаемом к этой книге, содержится текстовый файл с названием `win32api.txt`, в котором описаны операторы объявления и константы Windows API. Этот файл можно открыть в текстовом редакторе и оттуда скопировать соответствующие объявления в модуль VBA.

При работе с функциями API довольно часто встречаются системные ошибки во время тестирования, поэтому почаще сохраняйте проекты.

Если вы разрабатываете приложения, которые должны работать во всех версиях Excel, помните о некоторых потенциально “болезненных” вопросах совместимости, возникающих при вызове функций API. Например, приложение, разрабатываемое для Excel 97 или более поздней версии, которое использует функции API, не сможет быть запущено в Excel 5 — даже если вы сохраните рабочую книгу в формате Excel 5 (так как Excel 5 является 16-битовым приложением). Версии Excel 97 и выше — 32-битовые приложения. Подробнее об этой проблеме и способах ее предотвращения рассказано в главе 26.

Глава 11

Примеры и методы программирования на VBA

В ЭТОЙ ГЛАВЕ...

Изучение любого предмета, в том числе и программирования, можно сделать более эффективным, приведя ряд примеров. Практический подход используется и программистами на VBA. Хорошо продуманный пример, как правило, передает определенную идею лучше, чем описание теории, которая лежит в его основе. Поэтому я не ставил своей целью создать справочник, в котором скрупулезно описаны все нюансы работы с VBA. Вместо этого я подготовил ряд примеров, демонстрирующих полезные приемы программирования в Excel.

- Примеры использования VBA для работы с диапазонами
- Примеры использования VBA для управления рабочими книгами и листами
- Пользовательские функции, используемые в процедурах VBA и формулах рабочего листа
- Методы и способы написания программ на VBA
- Примеры использования функций Windows API

В предыдущих главах этой части был предоставлен вводный теоретический материал, кроме того, в электронной справочной системе можно найти все детальные сведения, которые в книге опущены. В этой главе приведены примеры решения практических задач, благодаря которым вы сможете углубить свои познания в области VBA.

Примеры, приведенные в данной главе, можно разделить на шесть категорий.

- Работа с диапазонами.
- Управление рабочими книгами и листами.
- Методы программирования на VBA.
- Функции, используемые в процедурах VBA.
- Функции, применяемые в формулах рабочего листа.
- Вызов функций Windows API.

ПЕРЕКРЕСТНАЯ ССЫЛКА

В последующих главах этой книги будут представлены частные примеры управления диаграммами, сводными таблицами, событиями, формами и т.п.

Использование примеров данной главы

Не все примеры в настоящей главе представляют независимые программы. Однако они являются исполняемыми процедурами, которые вы можете применять в собственных приложениях.

Настоятельно рекомендуем в процессе чтения данной главы работать за компьютером. Вы сможете изменять примеры и наблюдать, что же происходит. Этот практический опыт гарантированно даст вам больше, чем чтение справочников.

Работа с диапазонами

Примеры данного раздела демонстрируют принципы управления диапазонами на рабочих листах с помощью VBA.

Примеры из этого раздела можно найти на прилагаемом к книге компакт-диске.

Копирование диапазона

Функция записи макросов Excel используется не столько для создания хорошего кода, сколько для поиска названий необходимых объектов, методов и свойств. Программа, которая получена в результате записи макросов, не всегда самая эффективная, но обычно она предоставляет немало полезных сведений.

Например, при записи простой операции копирования и вставки мною получено пять строк VBA-кода.

```
Sub Macro1()
  Range("A1").Select
  Selection.Copy
  Range("B1").Select
  ActiveSheet.Paste
  Application.CutCopyMode = False
End Sub
```

Обратите внимание, что данная программа выделяет ячейки. Однако в VBA для работы с объектом не обязательно его выделять. Вы бы никогда не узнали об этом важном моменте, если бы копировали показанный выше код макроса, где в двух строках содержится метод `Select`. Данную процедуру можно заменить значительно более простой — применить метод `Copy`, который использует аргумент, представляющий адрес места вставки копируемого диапазона.

```
Sub CopyRange()
  Range("A1").Copy Range("B1")
End Sub
```

В обоих макросах предполагается, что рабочий лист является активным, и операция происходит на активном рабочем листе. Чтобы скопировать диапазон на другой рабочий лист или в другую книгу, необходимо задать ссылку. В следующем примере диапазон из листа `Лист1` книги `File.xls` копируется на лист `Лист2` книги `File2.xls`. Так как ссылки заданы правильно, пример работает независимо от того, какая рабочая книга активна.

```
Sub CopyRange2()
  Workbooks("File1.xls").Sheets("Лист1").Range("A1").Copy _
   Workbooks("File2.xls").Sheets("Лист2").Range("A1")
End Sub
```

Еще одним подходом к решению этой задачи является использование для представления диапазонов переменных объектов, как показано в следующем примере.

```
Sub CopyRange3()
  Set Rng1 = Workbooks("File1.xls"). _
   Sheets("Лист1").Range("A1")
  Set Rng2 = Workbooks("File2.xls"). _
   Sheets("Лист2").Range("A1")
  Rng1.Copy Rng2
End Sub
```

Понятно, что копирование не ограничивается единственной ячейкой за операцию. Например, следующая процедура копирует большой диапазон. Помните, что адрес места вставки определяется единственной ячейкой (представляющей верхний левый угол вставляемого диапазона).

```
Sub CopyRange4()
  Range("A1:C800").Copy Range("D1")
End Sub
```

Перемещение диапазона

Инструкции VBA для перемещения диапазона ячеек подобны инструкциям копирования диапазона (см. следующий пример). Разница заключается в том, что вместо метода `Copy` используется метод `Cut`. Обратите внимание, что для вставляемого диапазона необходимо задавать только адрес левой верхней ячейки.

Следующий пример демонстрирует операцию перемещения 18 ячеек (в диапазоне `A1:C6`) на новое место, начиная с ячейки `H1`.

```
Sub MoveRange()
  Range("A1:C6").Cut Range("H1")
End Sub
```

Копирование диапазона переменного размера

Во многих случаях необходимо скопировать диапазон ячеек, когда неизвестны точные размеры диапазона (количество столбцов и строк). Например, вы управляете рабочей книгой, в которой фиксируется объем продаж за неделю. Количество строк еженедельно меняется по мере добавления новых данных.

На рис. 11.1 показан стандартный вид рабочего листа. Диапазон состоит из нескольких строк, количество строк изменяется каждую неделю. Так как вы не можете знать точный размер диапазона в определенный момент времени, работа по написанию макроса, копирующего данный диапазон, несколько усложняется.



	A	B	C	D	E	F
1	Неделя	Объем продаж	Новые покупатели			
2	1	12,343	42			
3	2	9,832	12			
4	3	13,982	45			
5	4	13,722	21			
6	5	7,82	18			
7	6	17,83	54			
8	7	14,984	33			
9						
10						
11						
12						
13						

Рис. 11.1. Этот диапазон может состоять из любого количества строк

Следующий макрос демонстрирует, как скопировать данный диапазон из листа `Лист1` на лист `Лист2` (начиная с ячейки `A1`). В данном случае используется свойство `CurrentRegion`, возвращающее объект `Range`, который соответствует прямоугольнику ячеек вокруг заданной ячейки (в данном случае `A1`).

```
Sub CopyCurrentRegion2()
  Range("A1").CurrentRegion.Copy _
   Sheets("Лист2").Range("A1")
End Sub
```

Использование свойства `CurrentRegion` эквивалентно следующим действиям: выберите команду Правка⇨Перейти, щелкните на кнопке Выделить, выберите параметр текущую область. Чтобы увидеть принцип работы, включите запись макроса и выполните указанные действия. Как правило, значение свойства `CurrentRegion` состоит из прямоугольного диапазона ячеек, окруженных одной или более пустыми строками или столбцами.

Выделение или определение типов диапазонов

Зачастую работа, выполняемая в VBA, связана с управлением диапазонами — либо выделением диапазона, либо определением диапазона с целью выполнить определенные действия с ячейками.

В предыдущих версиях Excel запись макроса, выделяющего ячейки (как при нажатии <Ctrl+Shift+→ >), происходила достаточно бессистемно. Команда записи макросов в Excel 2003 обрабатывает такие типы выделений намного лучше, чем в предыдущих версиях. Однако рекомендуем тщательно проверять полученный при записи макросов код, чтобы убедиться, что выделение выполняется так, как это необходимо.

Кроме свойства `CurrentRegion` (описанного выше), вам необходимо научиться управлять методом `End` объекта `Range`. Метод `End` имеет один аргумент, определяющий направление, в котором увеличивается выделение ячеек. Оператор, представленный ниже, выделяет диапазон от активной ячейки до последней непустой ячейки внизу.

```
Range(ActiveCell, ActiveCell.End(xlDown)).Select
```

Как можно догадаться, три остальные константы имитируют комбинации клавиш при выделении в других направлениях: `xlUp` (вверх), `xlToLeft` (влево) и `xlToRight` (вправо).

Будьте внимательны при использовании метода `End`. Если активная ячейка находится на границе диапазона или диапазон содержит хотя бы одну пустую ячейку, метод `End` может не дать желаемых результатов.

На прилагаемом к книге компакт-диске находится рабочая книга, в которой содержится несколько популярных типов выделений ячеек. Открыв ее, вы увидите новое меню — Демонстрация выделения, которое содержит команды, позволяющие пользователю получать различные типы выделений (рис. 11.2).

Приведенный далее макрос взят из этой рабочей книги. Макрос `SelectCurrentRegion` имитирует нажатие клавиш <Ctrl+Shift+*>.

```
Sub SelectCurrentRegion()
  ActiveCell.CurrentRegion.Select
End Sub
```

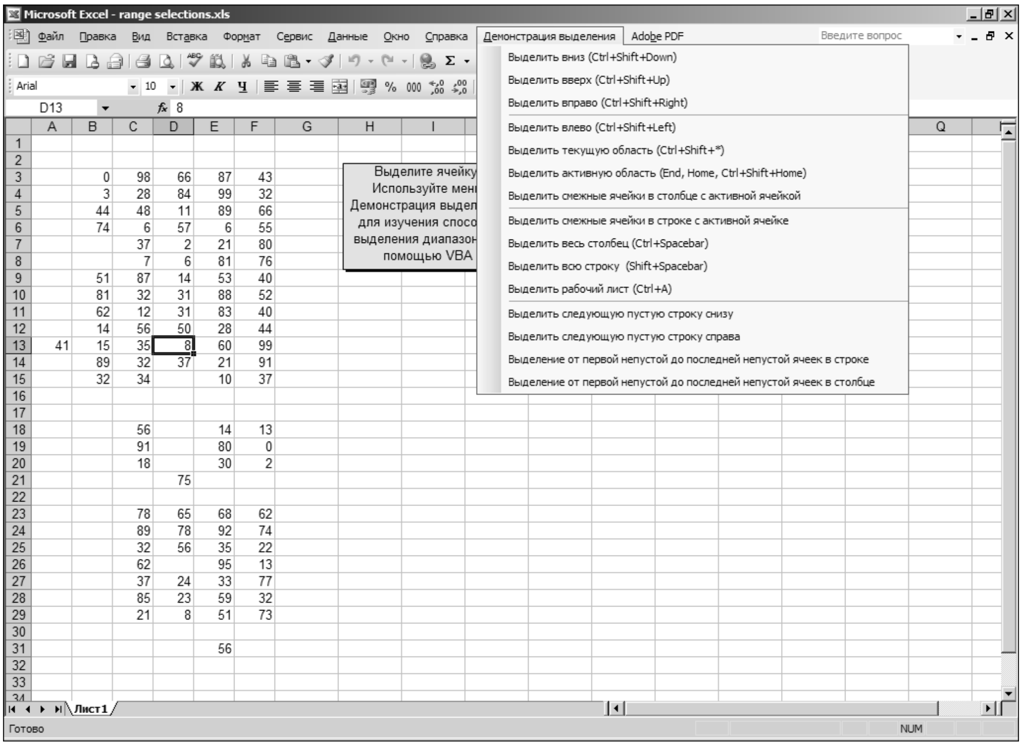


Рис. 11.2. Эта книга демонстрирует, как выделять диапазоны различной формы с помощью VBA

Часто для получения желаемого результата вам не придется выделять ячейки. Вместо этого необходимо выполнить над ними определенное действие (например, отформатировать). Процедуры выделения ячеек можно легко изменить. Процедура, показанная ниже, была получена на основе процедуры `SelectCurrentRegion` — она не выделяет ячейки, а создает объект `Range` и применяет к нему форматирование. Другие процедуры рабочей книги, представленные на компакт-диске, можно модифицировать таким же образом.

```
Sub FormatCurrentRegion()
  Set WorkRange = ActiveCell.CurrentRegion
  WorkRange.Font.Bold = True
End Sub
```

Советы по работе с диапазонами

При работе с диапазонами необходимо помнить о следующих моментах.

- Для управления диапазоном его не требуется выделять в программе.
- Если в коде выделяется диапазон, то соответствующий рабочий лист должен быть активным. Чтобы активизировать конкретный лист, используйте метод `Activate` коллекции `Worksheets`.
- Команда записи макросов не всегда генерирует самый эффективный код. Зачастую после записи макроса код необходимо отредактировать, чтобы сделать его более эффективным.

- Используйте в программе VBA именованные диапазоны. Например, лучше применить ссылку `Range("Total")`, чем ссылку `Range("D45")`. Во втором случае при добавлении строки над строкой 45 адрес ячейки изменится. Тогда следует изменить макрос, чтобы в нем использовался правильный адрес ячейки (`D46`).
- Если вы используете код, записанный при выделении диапазонов с помощью комбинации клавиш (например, <Ctrl+Shift+→ > для создания выделения до конца строки), то внимательно проверьте его. Иногда Excel использует абсолютные ссылки на выделенные ячейки.
- Перед выполнением макроса, который управляет каждой ячейкой в текущем диапазоне, рассмотрите ситуацию, когда пользователь выделил столбцы или строки целиком. В большинстве случаев нет необходимости, чтобы макрос просматривал каждую ячейку выделенного диапазона. На этот случай в макросе должен создаваться новый диапазон, который состоит только из непустых ячеек диапазона, выделенного пользователем.
- Excel позволяет выделять несколько несмежных диапазонов. Например, можно выделить диапазон, нажать <Ctrl> и выделить еще один диапазон. Такой диапазон можно определить в макросе и назначить для его управления дополнительные действия.

Запрос значения ячейки

Следующая процедура демонстрирует, как запросить у пользователя значение и вставить его в ячейку `A1` активного рабочего листа.

```
Sub GetValue1()
  Range("A1").Value = InputBox("Введите значение")
End Sub
```

На рис. 11.3 показано диалоговое окно ввода данных.



Рис. 11.3. Функция `InputBox` получает от пользователя значение, которое должно вставляться в ячейку

Однако в этой процедуре существует одна проблема. Если пользователь щелкнет на кнопке Отмена (Cancel) в окне ввода данных, то процедура удалит данные, которые находились в текущей ячейке. Модифицированная версия процедуры адекватно реагирует на щелчок на кнопке Отмена (Cancel) и не выполняет при этом никаких действий.

```
Sub GetValue2()
  UserEntry = InputBox("Введите значение")
  If UserEntry <> "" Then Range("A1").Value = UserEntry
End Sub
```

Во многих случаях следует проверить корректность данных, введенных пользователем. Например, необходимо обеспечить введение только чисел в диапазоне от 1 до 12. Далее представлен способ проверки данных, введенных пользователем. В приведенном ниже примере некорректные данные игнорируются, и окно запроса значения отображается снова. Этот цикл будет повторяться, пока пользователь не введет корректное значение или не щелкнет на кнопке Отмена (Cancel).

```
Sub GetValue3()
    Dim MinVal As Integer, MaxVal As Integer
    Dim UserEntry As String
    Dim Msg As String
    Dim IntEntry As Integer
```

```
    MinVal = 1
    MaxVal = 12
    Msg = "Введите значение от " & MinVal & " до " & MaxVal
    Do
        UserEntry = InputBox(Msg)
        If UserEntry = "" Then Exit Sub
        If IsNumeric(UserEntry) Then
            IntEntry = CInt(UserEntry)
            If IntEntry >= MinVal And IntEntry <= MaxVal Then
                Exit Do
            End If
        End If
        Msg = "Вы ввели НЕПРАВИЛЬНОЕ значение."
        Msg = Msg & vbNewLine
        Msg = Msg & "Введите значение от " & _
                     MinVal & " до " & MaxVal
    Loop
    ActiveSheet.Range("A1").Value = UserEntry
End Sub
```

Как видно из рис. 11.4, программа также изменяет отображаемое сообщение, если пользователь вводит некорректные данные.

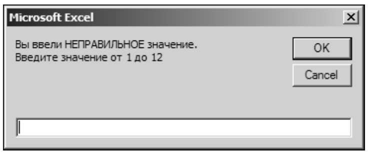


Рис. 11.4. Проверка корректности данных, введенных пользователем, с помощью функции VBA `InputBox`

Функция `InputBox` возвращает строковое значение, поэтому я использовал функцию `Val` для преобразования строки в число и только после этого применил оператор `If` для выполнения сравнения.

Ввод значения в следующую пустую ячейку

Достаточно часто требуется ввести значение в следующую пустую ячейку столбца или строки. В представленном далее примере пользователь должен ввести имя и значение, которые добавляются в следующую пустую строку (рис. 11.5).

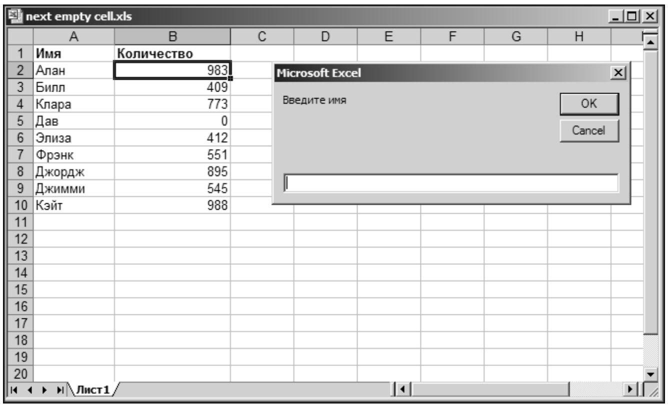


Рис. 11.5. Макрос вставки данных в следующую пустую строку рабочего листа

```
Sub GetData()
    Dim NextRow As Long
    Dim Entry1 As String, Entry2 As String
  Do
    NextRow = Range("A65536").End(xlUp).Row + 1
    Entry1 = InputBox("Введите имя")
    If Entry1 = "" Then Exit Sub
    Entry2 = InputBox("Введите сумму")
    If Entry2 = "" Then Exit Sub
    Cells(NextRow, 1) = Entry1
    Cells(NextRow, 2) = Entry2
  Loop
End Sub
```

Обратите внимание, что это бесконечный цикл. Для выхода из него (щелкните на кнопке Cancel) использовались операторы `Exit Sub`.

НА ЗАМЕТКУ: Чтобы упростить представленную процедуру, проверка данных не выполнялась.

Обратите внимание на оператор, который определяет значение переменной `NextRow`. Если вам трудно понять принцип его работы, то постарайтесь проанализировать содержимое ячейки: перейдите в ячейку `A65536` (последняя ячейка в столбце А), затем нажмите <End> и клавишу со стрелкой "вверх". При этом будет выделена последняя непустая ячейка в столбце А. Свойство `Row` возвращает номер этой строки; чтобы получить расположенную под ней строку (следующая пустая строка), к этому номеру прибавляется 1.

Стоит отметить, что в представленном приеме выделения следующей пустой строки существует такой нюанс: если столбец полностью пуст, то следующей пустой строкой будет считаться строка 2.

Приостановка макроса для получения диапазона, выделенного пользователем

Возможно, вам понадобится создать макрос, который останавливает выполнение кода, чтобы пользователь задал диапазон ячеек. Для этого воспользуйтесь функцией Excel `InputBox`.

Процедура, представленная ниже, демонстрирует, как приостановить макрос и позволить пользователю выбрать ячейку.

```
Sub GetUserRange()
  Dim UserRange As Range

  Output = 565
  Prompt = "Выберите ячейку для отображения результата"
  Title = "Выберите ячейку"

'  Отображение окна для ввода данных
  On Error Resume Next
  Set UserRange = Application.InputBox( _
    Prompt:=Prompt, _
    Title:=Title, _
    Default:=ActiveCell.Address, _
    Type:=8) 'Выделение диапазона
  On Error GoTo 0
'  Проверка, была ли нажата кнопка Отмена
  If UserRange Is Nothing Then
    MsgBox "Действие отменено"
```

```
  Else
    UserRange.Range("A1") = Output
  End If
End Sub
```

Окно ввода данных показано на рис. 11.6.

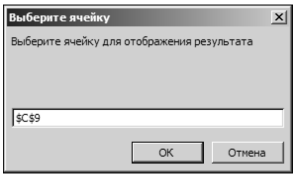


Рис. 11.6. Использование окна ввода данных с целью приостановить макрос

Ключевым моментом в этой процедуре является определение аргумента `Type` равным `8`. Кроме того, обратите внимание на использование оператора `On Error Resume Next`. Он игнорирует ошибку, происходящую по вине пользователя, который щелкает на кнопке Отмена. В таком случае переменная объекта `UserRange` не получает значения. В данном примере отображается окно сообщения с текстом “Действие отменено”. Если же пользователь щелкнет на кнопке ОК, то макрос продолжится. Строка `On Error Go To` указывает на переход к нормальной обработке ошибки.

Проверка корректного выделения диапазона необязательна. Excel позаботится об этом за вас.

ВНИМАНИЕ

Обязательно проверьте, включен ли параметр обновления экрана `ScreenUpdating`. В противном случае вы не сможете выбрать ячейку.

Подсчет выделенных ячеек

При работе с макросом, который обрабатывает выделенный диапазон ячеек, можно использовать свойство `Count`, чтобы определить, сколько ячеек содержится в выделенном диапазоне (или любом другом диапазоне). Например, следующий оператор демонстрирует окно сообщения, которое отображает количество ячеек в текущем выделенном диапазоне.

```
MsgBox Selection.Count
```

Если активный лист содержит диапазон с названием `data`, то следующий оператор присваивает количество ячеек в диапазоне `data` переменной с названием `CellCount`.

```
CellCount = Range("data").Count
```

Вы можете также определить, сколько строк или столбцов содержится в диапазоне. Следующее выражение вычисляет количество столбцов в выделенном в данный момент диапазоне.

```
Selection.Columns.Count
```

Для определения количества строк в диапазоне вы также можете использовать свойство `Rows`. Следующий оператор пересчитывает количество строк в диапазоне с названием `data` и присваивает это количество переменной `RowCount`.

```
RowCount = Range("data").Rows.Count
```

Определение типа выделенного диапазона

Excel поддерживает несколько типов диапазонов.

- Отдельная ячейка.
- Смежный диапазон ячеек.
- Один или несколько полных столбцов.
- Одна или несколько полных строк.
- Весь рабочий лист.
- Комбинация перечисленных выше типов (называемая множественным выделением).

В результате, когда процедура VBA обрабатывает выделенный диапазон, нельзя исходить из каких-либо предположений о типе этого диапазона.

В случае множественного выделения объект `Range` включает несмежные области. Чтобы определить, является ли выделение множественным, используется метод `Areas`, возвращающий коллекцию `Areas`. Эта коллекция представляет все диапазоны во множественном выделении.

Для определения того, содержатся ли в выделенном диапазоне множественные области, примените выражение, подобное следующему.

```
NumAreas = Selection.Areas.Count
```

Если переменная `NumAreas` содержит значение больше единицы, то выделение является множественным.

Процедура `AboutRangeSelection` использует функцию `AreaType`, показанную ниже.

```
Function AreaType(RangeArea As Range) As String
'   Возвращает тип диапазона
    Select Case True
        Case RangeArea.Cells.Count = 1
            AreaType = "Cell"
        Case RangeArea.Count = Cells.Count
            AreaType = "Worksheet"
        Case RangeArea.Rows.Count = Cells.Rows.Count
            AreaType = "Column"
        Case RangeArea.Columns.Count = Cells.Columns.Count
            AreaType = "Row"
        Case Else
            AreaType = "Block"
    End Select
End Function
```

Эта функция получает в качестве аргумента объект `Range` и возвращает одну из пяти строк, описывающих данную область: ячейка, рабочий лист, столбец, строка или блок. Функция использует конструкцию `Select Case` для определения одного из четырех выражений сравнения, которое имеет значение `True`. Например, если диапазон состоит из одной ячейки, функция возвращает `Cell`. Если количество ячеек в диапазоне равно количеству ячеек в рабочем листе, то функция возвращает `Worksheet`. Если количество строк в диапазоне равно количеству строк в рабочем листе, функция возвращает `Column`. Если количество столбцов в диапазоне равно количеству столбцов на рабочем листе, функция возвращает `Row`. Если ни одно из выражений `Case` не равно `True`, то функция возвращает строку `Block`.

Обратите внимание, что в сравнении не используются абсолютные числа. Например, вместо того чтобы использовать число 65536 для определения принадлежности диапазона к столбцу, применяется `Cells.Count`, что обеспечивает корректность выполнения функции даже в Excel 5 и Excel 97 (где лист состоит всего из 16384 строк).

Рабочая книга на прилагаемом компакт-диске содержит процедуру (с названием `AboutRangeSelection`), использующую функцию `AreaType` для отображения окна сообщения, в котором описан текущий выделенный диапазон (рис. 11.7). Понимание принципов работы этой процедуры позволит вам правильно управлять объектами `Range`.

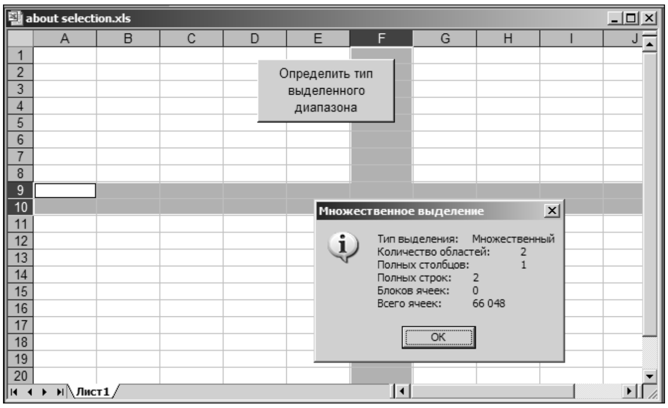


Рис. 11.7. Процедура `AboutRangeSelection` анализирует текущий выделенный диапазон

Возможно, вас удивит, что Excel позволяет создавать идентичные множественные выделения. Например, если, удерживая <Ctrl>, щелкнуть пять раз на ячейке `A1`, то выделение будет содержать пять идентичных областей. Это также учтено в процедуре `AboutRangeSelection`.

Просмотр выделенного диапазона

Вы можете столкнуться с проблемой, если необходимо создать макрос, который оценивает каждую ячейку в диапазоне и выполняет операцию, определенную заданному критерию. В листинге 11.1 приведен пример такого макроса: процедура `SelectiveColor1` устанавливает красный фон для всех ячеек, имеющих отрицательное значение. Цвет фона остальных ячеек не указывается.

Листинг 11.1. Красный фон ячеек с отрицательными значениями

```
Sub SelectiveColor1()
'   Задает красный фон для ячейки, если ее значение отрицательно
    If TypeName(Selection) <> "Range" Then Exit Sub
    Const REDINDEX = 3
    Application.ScreenUpdating = False
    For Each cell In Selection
        If cell.Value < 0 Then
          cell.Interior.ColorIndex = REDINDEX
        Else
          cell.Interior.ColorIndex = xlNone
        End If
    Next cell
End Sub
```

Процедура `SelectiveColor1` работает, но в ней имеется серьезный недостаток. К примеру, какой результат вы получите, если выделение состоит из полного столбца? Или десяти столбцов? Или всего рабочего листа? Пользователь, скорее всего, уснет, пока будут оценены все ячейки. Лучшее решение этой задачи (`SelectiveColor2`) представлено в листинге 11.2.

Листинг 11.2. Улучшение процедуры `SelectiveColor1` с учетом больших диапазонов, состоящих из нескольких столбцов

```
Sub SelectiveColor2()
'   Задает красный фон для ячейки, если ее значение отрицательно

    Dim FormulaCells As Range
    Dim ConstantCells As Range

    Const REDINDEX = 3

'   Игнорирует ошибки
    On Error Resume Next

    Application.ScreenUpdating = False

'   Создает подмножества первоначального выделения
    Set FormulaCells = Selection.SpecialCells _
      (xlFormulas, xlNumbers)
    Set ConstantCells = Selection.SpecialCells _
       (xlConstants, xlNumbers)

'   Обработка ячеек с формулами
    If Not FormulaCells Is Nothing Then
        For Each cell In FormulaCells
            If cell.Value < 0 Then _
              cell.Font.ColorIndex = REDINDEX
        Next cell
    End If

'   Обработка ячеек с константами
    If Not ConstantCells Is Nothing Then
        For Each cell In ConstantCells
            If cell.Value < 0 Then
              cell.Interior.ColorIndex = REDINDEX
            Else
              cell.Interior.ColorIndex = xlNone
            End If
        Next cell
    End If
End Sub
```

Эта процедура выполняет дополнительные действия, которые делают ее более эффективной. В ней используется метод `SpecialCells` для создания двух подмножеств текущего выделения: первое подмножество включает только ячейки с числовыми константами, а второе — только ячейки с формулами. Затем ячейки в этих подмножествах обрабатываются с помощью двух конструкций `For Each-Next`. В итоге оцениваются только непустые ячейки, что значительно ускоряет работу макроса.

НА ЗАМЕТКУ

Оператор `On Error` необходим, так как метод `SpecialCells` генерирует ошибку, если не находит в диапазоне ячеек указанного типа. Этот оператор также учитывает ситуации, когда процедура выполняется при невыделенном диапазоне.

Удаление всех пустых строк

Следующая процедура удаляет все пустые строки в активном рабочем листе. Она достаточно эффективна, так как не проверяет все без исключения строки, а просматривает только строки в так называемом "используемом диапазоне", определяемом с помощью свойства UsedRange объекта Worksheet.

```
Sub DeleteEmptyRows()
    Dim LastRow As Long, r As Long
    LastRow = ActiveSheet.UsedRange.Rows.Count
    LastRow = LastRow + ActiveSheet.UsedRange.Row - 1
    Application.ScreenUpdating = False
    For r = LastRow To 1 Step -1
        If Application.CountA(Rows(r)) = 0 Then Rows(r).Delete
    Next r
End Sub
```

Первый шаг — определить последнюю используемую строку и присвоить этот номер строки переменной LastRow. Это не так просто, как можно ожидать, поскольку текущий диапазон не обязательно начинается со строки 1. Следовательно, значение LastRow вычисляется таким образом: к найденному количеству строк используемого диапазона прибавляется номер первой строки текущего диапазона и вычитается 1.

В процедуре применена функция Excel СЧЕТЗ (COUNTA) для определения, является ли строка пустой. Если данная функция для конкретной строки возвращает 0, то эта строка пустая. Обратите внимание, что процедура просматривает строки снизу вверх и использует отрицательное значение шага в цикле For-Next. Это необходимо, поскольку при удалении все последующие строки перемещаются "вверх" в рабочем листе. Если в цикле просмотр выполняется бы сверху вниз, то счетчик цикла после удаления строки становится бы неправильным.

Определение диапазона, находящегося в другом диапазоне

Функция InRange, показанная ниже, имеет два аргумента, оба — объекты Range. Функция возвращает ИСТИНА, если первый диапазон содержится во втором.

```
Function InRange(rng1, rng2) As Boolean
'   Возвращает True, если rng1 является подмножеством rng2
    InRange = False
    If rng1.Parent.Parent.Name = rng2.Parent.Parent.Name Then
        If rng1.Parent.Name = rng2.Parent.Name Then
            If Union(rng1, rng2).Address = rng2.Address Then
                InRange = True
            End If
        End If
    End If
End Function
```

Возможно, функция InRange кажется сложнее, чем того требует ситуация, поскольку в коде должна быть реализована проверка принадлежности двух диапазонов одной и той же книге и рабочему листу. Обратите внимание, что в процедуре используется свойство Parent, которое возвращает объект-контейнер заданного объекта. Например, следующее выражение возвращает название листа для объекта rng1.

```
rng1.Parent.Name
```

Представленное далее выражение возвращает имя рабочей книги для объекта rng1.

```
rng1.Parent.Parent.Name
```

Функция VBA Intersection возвращает объект Range, представляющий пересечение двух объектов Range. Это пересечение содержит общие ячейки двух заданных диапазонов. Если пересечение двух диапазонов совпадает со вторым диапазоном, то это означает, что первый диапазон полностью содержится во втором.

Определение типа данных ячейки

Excel имеет ряд встроенных функций, которые могут помочь определить тип данных, содержащихся в ячейке. Это функции ЕНЕТЕКСТ (ISTEXT), ЕЛОГИЧ (ISLOGICAL) и ЕОШИБКА (ISERROR). Кроме того, VBA поддерживает функции IsEmpty, IsDate и IsNumeric.

Ниже описана функция CellType, которая принимает аргумент-диапазон и возвращает строку (Пусто, Текст, Булево выражение, Ошибка, Дата, Время или Значение), описывающую тип данных левой верхней ячейки этого диапазона. Такую функцию можно использовать в формуле рабочего листа или вызвать из другой процедуры VBA.

```
Function CellType(Rng)
'   Возвращает тип верхней левой ячейки
'   в диапазоне
    Application.Volatile
    Set Rng = Rng.Range("A1")
    Select Case True
        Case IsEmpty(Rng)
            CellType = "Пусто"
        Case WorksheetFunction.IsText(Rng)
            CellType = "Текст"
        Case WorksheetFunction.IsLogical(Rng)
            CellType = "Булево выражение"
        Case WorksheetFunction.IsErr(Rng)
            CellType = "Ошибка"
        Case IsDate(Rng)
            CellType = "Дата"
        Case InStr(1, Rng.Text, ":") <> 0
            CellType = "Время"
        Case IsNumeric(Rng)
            CellType = "Значение"
    End Select
End Function
```

Обратите внимание на использование оператора Set Rng. Функция CellType получает аргумент-диапазон произвольного размера, но этот оператор указывает, что функция оперирует только левой верхней ячейкой диапазона (представленной переменной TheCell).

Чтение и запись диапазонов

Многие задачи, выполняемые в электронных таблицах, связаны с переносом значений из массива в диапазон ячеек или из диапазона в массив. Следует отметить, что Excel получает данные из диапазонов значительно быстрее, чем записывает их. Процедура WriteReadRange, показанная в листинге 11.3, демонстрирует относительную скорость записи и чтения диапазона.

Эта процедура создает массив и затем использует циклы For-Next для записи данного массива в диапазон и считывания данных обратно в массив. С помощью функции Timer вычисляется время, необходимое для каждой операции.

Листинг 11.3. Тестирование скорости выполнения операций чтения и записи в диапазоне

```
Sub WriteReadRange()
    Dim MyArray()
    Dim Time1 As Date
    Dim NumElements As Long, i As Long
    Dim WriteTime As String, ReadTime As String
    Dim Msg As String

    NumElements = 60000
    ReDim MyArray(1 To NumElements)

'   Заполнение массива
    For i = 1 To NumElements
        MyArray(i) = i
    Next i

'   Запись массива в диапазон
    Time1 = Timer
    For i = 1 To NumElements
        Cells(i, 1) = MyArray(i)
    Next i
    WriteTime = Format(Timer - Time1, "00:00")

'   Считывание диапазона в массив
    Time1 = Timer
    For i = 1 To NumElements
        MyArray(i) = Cells(i, 1)
    Next i
    ReadTime = Format(Timer - Time1, "00:00")

'   Отображение результатов
    Msg = "Запись: " & WriteTime
    Msg = Msg & vbCrLf
    Msg = Msg & "Чтение: " & ReadTime
    MsgBox Msg, vbOKOnly, NumElements & " элементов"
End Sub
```

В моей системе для записи массива из 60000 элементов в диапазон потребовалось 15 секунд, и только 4 секунды ушло на то, чтобы занести этот диапазон обратно в массив.

Более эффективный способ записи в диапазон

В примере предыдущего раздела для перемещения содержимого массива в диапазон используется цикл `For-Next`. В данном разделе показан более эффективный способ выполнения этой операции.

Начнем с примера в листинге 11.4, в котором продемонстрирован наиболее очевидный (но не самый эффективный) способ заполнения диапазона. В этом примере для вставки значений в диапазон используется цикл `For-Next`.

Листинг 11.4. Прямолинейное заполнение диапазона

```
Sub LoopFillRange()
'   Заполнение диапазона в цикле

    Dim CellsDown As Long, CellsAcross As Integer
    Dim CurrRow As Long, CurrCol As Integer
    Dim StartTime As Date
    Dim CurrVal As Long

'   Получение размеров
```

```
    CellsDown = Val(InputBox("Сколько ячеек в высоту?"))
    CellsAcross = Val(InputBox("Сколько ячеек в ширину?"))

'   Запись момента начала
    StartTime = Timer

'   Просмотр ячеек и вставка значений
    CurrVal = 1
    Application.ScreenUpdating = False
    For CurrRow = 1 To CellsDown
        For CurrCol = 1 To CellsAcross
            ActiveCell.Offset(CurrRow - 1, _
            CurrCol - 1).Value = CurrVal
            CurrVal = CurrVal + 1
        Next CurrCol
    Next CurrRow

'   Отображение времени выполнения операции
    Application.ScreenUpdating = True
    MsgBox Format(Timer - StartTime, "00.00") & " seconds"
End Sub
```

Пример в листинге 11.5 демонстрирует самый эффективный способ получения того же результата. Программа вставляет значения в массив и использует всего один оператор для перенесения содержимого массива в диапазон.

Листинг 11.5. Быстрый перенос массива в диапазон

```
Sub ArrayFillRange()
'   Заполнение диапазона путем переноса массива

    Dim CellsDown As Long, CellsAcross As Integer
    Dim i As Long, j As Integer
    Dim StartTime As Date
    Dim TempArray() As Long
    Dim TheRange As Range
    Dim CurrVal As Long

'   Получение размеров
    CellsDown = Val(InputBox("Сколько ячеек в высоту?"))
    CellsAcross = Val(InputBox("Сколько ячеек в ширину"))

'   Запись момента начала
    StartTime = Timer

'   Изменение размерности временного массива
    ReDim TempArray(1 To CellsDown, 1 To CellsAcross)

'   Определение диапазона на рабочем листе
    Set TheRange = ActiveCell.Range(Cells(1, 1), _
        Cells(CellsDown, CellsAcross))

'   Заполнение временного массива
    CurrVal = 0
    Application.ScreenUpdating = False
    For i = 1 To CellsDown
        For j = 1 To CellsAcross
            TempArray(i, j) = CurrVal + 1
            CurrVal = CurrVal + 1
        Next j
    Next i

'   Перенос временного массива на рабочий лист
    TheRange.Value = TempArray
```

```
'   Отображение времени выполнения операции
    Application.ScreenUpdating = True
    MsgBox Format(Timer - StartTime, "00.00") & " секунд"
End Sub
```

Например, в моей системе на заполнение массива размером 500×256 ячеек (128000 ячеек) методом цикла уходит 202,34 секунды. Метод перенесения массива потребовал только 0,77 секунды для получения тех же самых результатов — более чем в 250 раз быстрее! Мораль? Если необходимо переносить большие объемы данных на лист Excel, по возможности избегайте циклов.

Перенесение одномерных массивов

В примере предыдущего раздела рассматривался двухмерный массив, который прекрасно подходит для управления прямоугольными диапазонами (содержащими несколько строк и столбцов).

При переносе одномерного массива диапазон должен быть горизонтальным — т.е. это должна быть строка длительностью в несколько *столбцов*. Если же необходимо использовать вертикальный диапазон, сначала следует транспонировать массив. Для этого используйте функцию Excel `ТРАНСП` (`TRANSPOSE`). В следующем примере 100-элементный массив вставляется в вертикальный диапазон на рабочем листе (`A1:A100`).

```
Range(A1:A100).Value = _
  Application.WorksheetFunction.Transpose(MyArray)
```

Перенесение диапазона в массив Variant

В настоящем разделе рассматривается еще один способ управления данными Excel в VBA. В примере, показанном ниже, диапазон ячеек переносится в двухмерный массив `Variant`. Затем в окнах сообщений отображаются границы обоих размерностей массива `Variant`.

```
Sub RangeToVariant()
  Dim x As Variant
  x = Range("A1:L600")
  MsgBox UBound(x, 1)
  MsgBox UBound(x, 2)
End Sub
```

В данном примере в первом окне сообщения отображается 600 (количество строк в массиве), а во втором окне сообщения — 12 (количество столбцов). Вы увидите, что перенос данных диапазона в массив `Variant` происходит за долю секунды.

Следующий пример считывает диапазон в массив `Variant`, выполняет простую операцию умножения над каждым элементом массива и перемещает массив `Variant` обратно в диапазон.

```
Sub RangeToVariant2()
    Dim UserRange As Range
    Dim x As Variant
    Dim r As Long, c As Integer

    Set UserRange = Range("A1:L600")

'   Чтение данных
    x = Range("A1:L50")

'   Просмотр массива
    For r = 1 To UBound(x, 1)
        For c = 1 To UBound(x, 2)
'           Multiply by 2
```

```
            x(r, c) = x(r, c) * 2
        Next c
    Next r

'   Перемещение массива обратно на рабочий лист
    Range("A1:L50") = x
End Sub
```

Данная процедура работает очень быстро.

Выделение максимального значения в диапазоне

Процедура GoToMax в листинге 11.6 активизирует ячейку рабочего листа, содержащую максимальное значение. Процедура определяет максимальное значение в выделенном диапазоне; если выделена одна ячейка, то определяется максимальное значение на всем листе. Для определения адреса необходимой ячейки и ее выделения используется метод Find.

Листинг 11.6. Переход к ячейке, содержащей наибольшее значение

```
Sub GoToMax()
'   Активизирует ячейку с наибольшим значением
    Dim WorkRange as Range
    Dim MaxVal as Double
 '   Выход, если диапазон не выбран
    If TypeName(Selection) <> "Range" Then Exit Sub

'   Если выбрана одна ячейка, поиск по всему листу;
'   в противном случае - поиск в выделенном диапазоне
    If Selection.Count = 1 Then
        Set Workrange = Cells
    Else
        Set Workrange = Selection
    End If

'   Определение максимального значения
    MaxVal = Application.Max(Workrange)

'   Поиск и выделение ячейки с максимальным значением
    On Error Resume Next
    Workrange.Find(What:=MaxVal, _
        After:=Workrange.Range("A1"), _
        LookIn:=xlValues, _
        LookAt:=xlPart, _
        SearchOrder:=xlByRows, _
        SearchDirection:=xlNext, MatchCase:=False _
        ).Select
    If Err <> 0 Then MsgBox "Максимальное значение не найдено: " _
     & MaxVal
End Sub
```

Возможно, вы заметили, что аргументы метода Find совпадают с элементами управления в диалоговом окне Excel **Найти и заменить**.

Выделение всех ячеек с определенным форматированием

Пример данного раздела демонстрирует использование объекта FindFormat для поиска и выделения всех ячеек на рабочем листе, которые содержат указанное форматирование. Когда эти ячейки выделены, над ними можно выполнить любую операцию — изменить форматирование, удалить и т.п. На рис. 11.8 показан пример.

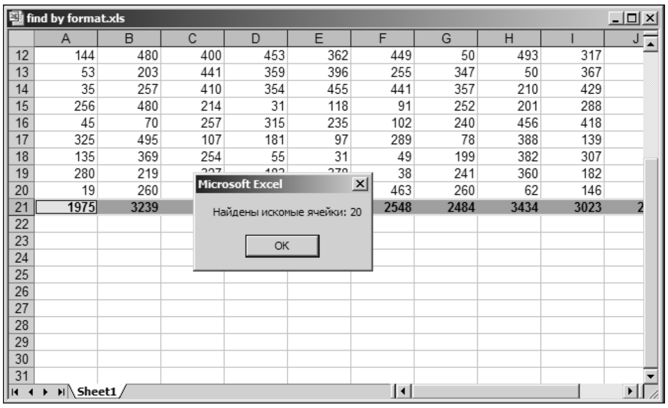


Рис. 11.8. Выделение всех ячеек с одинаковым форматированием

НОВИНКА

Свойство `FindFormat` впервые появилось в Excel 2002. Следовательно, данная процедура не будет работать в более ранних версиях Excel.

Процедура `SelectByFormat` выглядит следующим образом.

```
Sub SelectByFormat()
'   Выделяет ячейки на основе их форматирования

'   Используется Excel 2002 или выше
    If Val(Application.Version) < 10 Then
        MsgBox "Необходима Excel 2002 или выше"
        Exit Sub
    End If

    Dim FirstCell As Range, FoundCell As Range
    Dim AllCells As Range

'   Определение форматирования
    With Application.FindFormat
        .Clear
        .Interior.ColorIndex = 6 'желтый
        .Font.Bold = True
    End With

'   Поиск первой соответствующей ячейки
    Set FirstCell = Cells.Find(What:="", SearchFormat:=True)

'   Если ячейка не найдена, выход
    If FirstCell Is Nothing Then
        MsgBox "Ячейки указанного формата не найдены"
        Exit Sub
    End If

'   Инициализация AllCells
    Set AllCells = FirstCell
    Set FoundCell = FirstCell

'   Просмотр, пока не будет найдена ячейка FirstCell
    Do
        Set FoundCell = Cells.FindNext(After:=FoundCell)
        Set AllCells = Union(FoundCell, AllCells)
        If FoundCell.Address = FirstCell.Address Then Exit Do
    Loop
```

```
'   Выделение найденных ячеек и сообщение пользователю
    AllCells.Select
    MsgBox "Найдено ячеек: " & AllCells.Count
End Sub
```

Процедура начинается с задания свойств объекта `FindFormat`. В данном примере искомое форматирование определяется двумя компонентами: желтый фон и полужирный шрифт текста. Конечно, вы можете изменить эти настройки в коде на другие.

Метод `Find` используется для поиска первой ячейки, соответствующей критерию. Аргумент `What` метода `Find` вначале содержит пустую строку, так как поиск определяется форматированием, а не содержимым ячейки. Кроме того, аргумент `SearchFormat` имеет значение `True`, поскольку действительно выполняется поиск форматирования, а не значения.

Если указанное форматирование не найдено, пользователь получает сообщение, и программа заканчивает свою работу. В противном случае найденная ячейка присваивается переменной объекта `AllCells` (где хранятся все найденные ячейки). Цикл использует метод `FindNext` для продолжения поиска, который останавливается лишь тогда, когда снова будет найдена первая ячейка. Наконец, все найденные ячейки на рабочем листе выделяются, и пользователь получает сообщение о количестве найденных ячеек.

В представленной процедуре не производится поиск ячеек, получающих форматирование в результате применения команды условного форматирования Excel.

Управление рабочими книгами и листами

Приводимые в этом разделе примеры демонстрируют различные способы использования VBA для управления рабочими книгами и листами.

Примеры настоящего раздела содержатся на прилагаемом к книге компакт-диске.

Сохранение всех рабочих книг

Следующая процедура циклически просматривает все рабочие книги в коллекции `Workbooks` и сохраняет каждый файл, который сохранялся ранее.

```
Public Sub SaveAllWorkbooks()
  Dim Book As Workbook
  For Each Book In Workbooks
    If Book.Path <> "" Then Book.Save
  Next Book
End Sub
```

Обратите внимание, как используется свойство `Path`. Если для какой-либо рабочей книги свойство `Path` не задано, то это означает, что файл еще не сохранялся (это новая рабочая книга). Данная процедура игнорирует такие рабочие книги и сохраняет только те из них, у которых свойство `Path` имеет значение.

Сохранение и закрытие всех рабочих книг

Следующая процедура циклически просматривает коллекцию `Workbooks`. Программа сохраняет и закрывает все рабочие книги.

```
Sub CloseAllWorkbooks()
  Dim Book As Workbook
  For Each Book In Workbooks
    If Book.Name <> ThisWorkbook.Name Then
      Book.Close savechanges:=True
    End If
  Next Book
  ThisWorkbook.Close savechanges:=True
End Sub
```

Обратите внимание, что процедура использует оператор `If` для определения, содержит ли данная рабочая книга текущий исполняемый код. Это необходимо, так как при закрытии рабочей книги, содержащей процедуру, программа автоматически завершает свое выполнение, причем остальные рабочие книги не будут сохранены и закрыты.

Доступ к свойствам рабочей книги

Команда Excel Файл⇨Свойства отображает диалоговое окно, содержащее информацию об активной рабочей книге. Вы можете обратиться к этим свойствам с помощью VBA. Например, представленная далее процедура отображает дату и время последнего сохранения активной рабочей книги.

```
Sub LastSaved()
    Dim SaveTime As String
    On Error Resume Next
    SaveTime = ActiveWorkbook. _
      BuiltinDocumentProperties("Last Save Time").Value
    If SaveTime = "" Then
        MsgBox ActiveWorkbook.Name & " не была сохранена"
    Else
        MsgBox "Сохранено: " & SaveTime, , ActiveWorkbook.Name
    End If
End Sub
```

Если рабочая книга не сохранена, то при попытке получить доступ к свойству `Last Save Time` будет получена ошибка. Оператор `On Error` пропускает ошибку. Структура `If-Then-Else` проверяет значение переменной `SaveTime` и отображает соответствующее сообщение. Если переменная пуста, это означает, что файл не был сохранен. На рис. 11.9 показан пример выполнения этой процедуры.

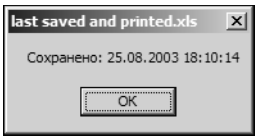


Рис. 11.9. Отображение даты и времени сохранения рабочей книги

НА ЗАМЕТКУ: Существует еще несколько встроенных свойств, к которым можно обратиться с помощью VBA, но эти немногочисленные свойства не относятся к опциям Excel. Полный список встроенных свойств вы можете найти в справочной системе.

Синхронизация рабочих листов

Если вы работаете с рабочими книгами, состоящими из нескольких листов, то, вероятно, знаете, что Excel не может "синхронизировать" листы в рабочей книге. Другими словами, не существует автоматического способа сделать так, чтобы все листы имели одинаковые выделенные диапазоны и верхние левые ячейки. Макрос VBA, показанный ниже, берет за основу активный рабочий лист и выполняет следующие действия со всеми остальными рабочими листами в книге.

- Выделяет тот же диапазон, что и в активном листе.
- Задает ту же левую верхнюю ячейку, что и на активном листе.

Ниже приведен листинг данного макроса.

```
Sub SynchSheets()
'   Дублирует активный диапазон и верхнюю левую ячейку
'   активного листа во всех рабочих листах
    If TypeName(ActiveSheet) <> "Worksheet" Then Exit Sub
    Dim UserSheet As Worksheet, sht As Worksheet
    Dim TopRow As Long, LeftCol As Integer
    Dim UserSel As String

    Application.ScreenUpdating = False

'   Сохранение текущего листа
    Set UserSheet = ActiveSheet

'   Сохранение информации из активного листа
    TopRow = ActiveWindow.ScrollRow
    LeftCol = ActiveWindow.ScrollColumn
    UserSel = ActiveWindow.RangeSelection.Address

'   Просмотр рабочих листов
    For Each sht In ActiveWorkbook.Worksheets
        If sht.Visible Then 'пропустить скрытые листы
            sht.Activate
            Range(UserSel).Select
            ActiveWindow.ScrollRow = TopRow
            ActiveWindow.ScrollColumn = LeftCol
        End If
    Next sht

'   Переход к первоначальной позиции
    UserSheet.Activate
    Application.ScreenUpdating = True
End Sub
```

Методы программирования на VBA

Примеры в этом разделе иллюстрируют часто используемые приемы VBA, которые вы можете использовать в собственных проектах.

НА КОМПАКТ-ДИСКЕ

Примеры, приведенные в данном разделе, можно найти на прилагаемом к книге компакт-диске.

Переключение значения свойства Boolean

Свойство Boolean — это логическое свойство, принимающее одно из двух значений: True (ИСТИНА) или False (ЛОЖЬ). Самый простой способ изменить логическое свойство — использовать оператор Not, как показано в следующем примере, в котором активизируется свойство переноса по словам WrapText в выделенном диапазоне ячеек.

```
Sub ToggleWrapText()
'  Управляет переносом слов в выделенных ячейках
  If TypeName(Selection) = "Range" Then
   Selection.WrapText = Not ActiveCell.WrapText
  End If
End Sub
```

Обратите внимание, что за основу взята активная ячейка. Когда диапазон выделен и значения свойств в разных ячейках неодинаковы (например, в некоторых ячейках шрифт полужирный, а в других — нет), то диапазон считается *смешанным*, и Excel использует в качестве базового значение свойства активной ячейки. Если, например, активная ячейка имеет полужирный шрифт, то начертание текста в выделенных ячейках при щелчке на кнопке Полужирный на панели инструментов будет обычным. Эта простая процедура имитирует поведение инструмента Excel.

Отметьте, что процедура использует функцию TypeName для проверки, является ли выделенный объект диапазоном. Если это не так, то ничего не происходит.

Оператор Not можно использовать для переключения значений многих свойств. Например, для отображения заголовков строк и столбцов в рабочем листе примените следующую команду.

```
ActiveWindow.DisplayHeadings = Not _
 ActiveWindow.DisplayHeadings
```

Для отображения линий сетки на активном листе воспользуйтесь таким кодом.

```
ActiveWindow.DisplayGridlines = Not _
 ActiveWindow.DisplayGridlines
```

Определение количества страниц для печати

Если вам необходимо определить количество страниц для печати, то можете использовать команду Excel Предварительный просмотр и посчитать количество страниц, которое отображается в нижней части экрана. Следующая процедура VBA вычисляет количество страниц для печати на активном листе путем подсчета горизонтальных и вертикальных разрывов страницы.

```
Sub PageCount()
    MsgBox (ActiveSheet.HPageBreaks.Count + 1) * _
      (ActiveSheet.VPageBreaks.Count + 1)
End Sub
```

Представленная ниже процедура VBA циклически просматривает все листы в активной рабочей книге и отображает общее количество страниц для печати.

```
Sub ShowPageCount()
    Dim PageCount As Integer
    Dim sht As Worksheet
    PageCount = 0
    For Each sht In Worksheets
       PageCount = PageCount + (sht.HPageBreaks.Count + 1) * _
        (sht.VPageBreaks.Count + 1)
    Next sht
    MsgBox "Всего страниц = " & PageCount
End Sub
```

Отображение даты и времени

Если вы разбираетесь в системе, используемой в Excel для хранения дат и времени, то у вас не возникнет проблем при работе со значениями дат и времени в процедурах VBA.

Процедура `DateAndTime` отображает окно сообщения с текущей датой и временем (рис. 11.10). В этом примере в строке заголовка окна сообщения представлено пользовательское сообщение.

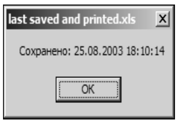


Рис. 11.10. Окно сообщения, отображающее дату и время

Процедура, приведенная в листинге 11.7, использует в качестве аргумента функции `Format` функцию `Date`. В результате строка с датой будет представлена в удобном для восприятия формате. Тот же прием применяется для задания формата времени.

Листинг 11.7. Отображение текущей даты и текущего времени

```
Sub DateAndTime()
    TheDate = Format(Date, "Long Date")
    TheTime = Format(Time, "Medium Time")

'   Определение приветствия в зависимости от времени суток
    Select Case Time
        Case Is < TimeValue("12:00"): Greeting = "Доброе утро, "
        Case Is >= TimeValue("17:00"): Greeting = "Добрый день, "
        Case Else: Greeting = "Добрый вечер, "
    End Select

'   Присоединение к приветствию имени пользователя
    FullName = Application.UserName
    SpaceInName = InStr(1, FullName, " ", 1)

'   Обработка ситуации, когда в имени нет пробела
    If SpaceInName = 0 Then SpaceInName = Len(FullName)
    FirstName = Left(FullName, SpaceInName)
    Greeting = Greeting & FirstName

'   Отображение сообщения
    MsgBox TheDate & vbCrLf & TheTime, vbOKOnly, Greeting
End Sub
```

В данном примере использованы именованные форматы (“Long Date” и “Medium Time”) с целью обеспечить работоспособность макроса независимо от региональных настроек компьютера пользователя. Однако вы можете обратиться к другим форматам. Например, чтобы отобразить дату в формате `мм/дд/гг`, воспользуйтесь следующим оператором.

```
TheDate = Format(Date, "mm/dd/yy")
```

Чтобы построить в зависимости от времени суток приветствие, которое отображается в строке заголовка, используется конструкция `Select Case`. Значения времени задаются в VBA так же, как в Excel. Если время меньше 0,5 (полдень), то это утро. Если время больше 0,7083 (5 часов вечера), то это вечер. Все остальное время —

это день. Мы выбрали легкий способ и использовали функцию VBA `TimeValue`, которая возвращает значение времени из строки.

Следующие операторы определяют имя пользователя, указанное на вкладке Общие диалогового окна Параметры. Для нахождения первого пробела в имени пользователя использована функция VBA `InStr`. Когда я создавал рассматриваемую процедуру в первый раз, то не учел, что в имени пользователя пробел может отсутствовать. Поэтому, когда процедура была запущена в компьютере с именем пользователем *Nobody*, программа выдала ошибку — из чего следует, что нельзя предусмотреть все, и даже самые простые процедуры могут дать сбой. (Кстати говоря, если поле введения имени пользователя не заполнено, то Excel всегда использует значение *User*.) Решение этой проблемы состоит в следующем: присвойте переменной `SpaceInName` длину полного имени пользователя, тогда функция `Left` извлечет полное имя.

Функция `MsgBox` объединяет дату и время, но использует встроенную константу `vbCrLf` для вставки между ними разрыва строки. `vbOKOnly` — предопределенная константа, возвращающая 0; в результате окно сообщения содержит только кнопку OK. Последний аргумент — приветствие `Greeting`, составленное ранее в процедуре.

Получение списка шрифтов

Если вам необходимо познакомиться со списком всех установленных шрифтов, то помните, что в Excel нет прямого способа получить эту информацию. Вам потребуется читать названия шрифтов из элемента управления Шрифт на панели инструментов Форматирование.

Следующая процедура отображает список установленных шрифтов в столбце A активного рабочего листа. Используется метод `FontConrol` для обращения к элементу управления Шрифт на панели инструментов Форматирование. Если такой элемент управления не найден (например, пользователь его удалил), то создается временная панель инструментов `CommandBar`, на которую добавляется элемент управления Шрифт.

Дополнительную информацию о работе с элементами управления `CommandBar` вы найдете в главе 22.

```
Sub ShowInstalledFonts()
    Dim FontList As CommandBarControl
    Dim TempBar As CommandBar
    Dim i As Integer

    Set FontList = Application.CommandBars("Formatting"). _
       FindControl(ID:=1728)

'   Если элемент Шрифт не отображен, создается временная панель
    If FontList Is Nothing Then
        Set TempBar = Application.CommandBars.Add
        Set FontList = TempBar.Controls.Add(ID:=1728)
    End If

'   Помещение шрифтов в столбец A
    Range("A:A").ClearContents
    For i = 0 To FontList.ListCount - 1
        Cells(i + 1, 1) = FontList.List(i + 1)
    Next i

'   Удаление временной панели, если она существует
    On Error Resume Next
    TempBar.Delete
End Sub
```

Дополнительно вы можете отобразить имя каждого шрифта с помощью самого шрифта. Для этого добавьте следующее выражение в цикл `For-Next`.

```
Cells(i+1,1).Font.Name = FontList.List(i+1)
```

Будьте внимательны, использование большого количества шрифтов в рабочей книге приводит к загруженности ресурсов компьютера, что часто вызывает сбои в системе.

Сортировка массива

Несмотря на то, что в Excel существует встроенная команда сортировки ячеек, в VBA метод сортировки массивов не представлен. Один возможный, но достаточно неудобный вариант решить эту задачу — перенести массив в диапазон ячеек на рабочем листе, отсортировать данные с помощью команд Excel, а затем занести результат обратно в массив. Однако если в вашей программе имеет большое значение скорость выполнения операции, то лучше написать на VBA процедуру сортировки.

В данном разделе рассматривается несколько методов сортировки.

- *Сортировка на рабочем листе.* Массив переносится на рабочий лист Excel; диапазон на рабочем листе сортируется и переносится обратно в массив. Единственным аргументом этой процедуры является массив. Работа может производиться с массивами не более чем из 65536 элементов (количество строк в рабочем листе).
- *Пузырьковый метод* — довольно простой прием сортировки (он использовался в примере сортировки листов в главе 9). Его несложно запрограммировать, однако такой алгоритм сортировки не самый эффективный, особенно для большого количества элементов в массиве.
- *Быстрая сортировка.* Намного более быстрая процедура, чем пузырьковый алгоритм, но, чтобы в ней разобраться, потребуется время.
- *Метод пересчета.* Работает очень быстро, однако для его улучшения также потребуется время и определенные усилия.

На прилагаемом к книге компакт-диске содержится приложение рабочей книги, демонстрирующее эти методы сортировки. Такую рабочую книгу интересно протестировать на массивах разного размера.

На рис. 11.11 показано диалоговое окно для этого проекта. Результаты тестирования получены для семи массивов разного размера (от 100 и до 100000 элементов); элементами массивов выступали произвольные числа (типа `Double`).

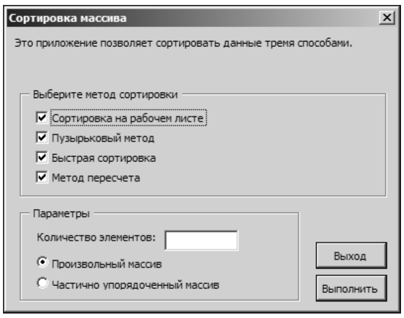


Рис. 11.11. Сравнение времени, необходимого для выполнения сортировки массивов различного размера

В табл. 11.1 представлены результаты теста. Выражение 0,00 означает, что сортировка произошла за время менее 0,01 секунды.

Таблица 11.1. Время сортировки в секундах для четырех алгоритмов, полученное для массивов с разным количеством элементов

Элементы массива	Сортировка на рабочем листе Excel	Пузырьковая сортировка VBA	Быстрая сортировка VBA	Сортировка методом пересчета VBA
100	0,05	0,00	0,05	0,00
500	0,06	0,11	0,05	0,00
1000	0,11	0,44	0,11	0,00
5000	0,55	8,89	0,77	0,05
10000	1,16	31,69	1,75	0,06
50000	6,98	788,62	10,21	0,22
100000	–	–	20,60	0,44

Как видно, на алгоритмы не оказывает особого влияния то, каким образом отсортированы элементы массива: находятся они в произвольном порядке или частично отсортированы.

Алгоритм сортировки на рабочем листе поразительно быстрый, принимая во внимание то, что массив переносится на лист, сортируется и затем переносится обратно в массив. Если массив практически отсортирован, то метод сортировки на рабочем листе выполняется довольно эффективно.

Обработка последовательности файлов

Одной из главных причин использования макросов является многократное повторение определенной операции. Пример в листинге 11.8 показывает, как выполнить макрос в нескольких разных файлах, сохраненных на диске. Этот пример, который призван помочь вам написать собственную программу выполнения этой задачи, запрашивает у пользователя сведения о файле и обрабатывает соответствующие запросу рабочие книги. В рассматриваемом случае обработка состоит из импорта файла и ввода ряда формул суммирования, описывающих данные в файле.

Листинг 11.8. Макрос, обрабатывающий несколько файлов на диске

```
Sub BatchProcess()
    Dim FS As FileSearch
    Dim FilePath As String, FileSpec As String
    Dim i As Integer

'   Определение пути и сведений о файле
    FilePath = ThisWorkbook.Path & "\"
    FileSpec = "text??.txt"

'   Создание объекта FileSearch
    Set FS = Application.FileSearch
    With FS
        .NewSearch
        .LookIn = FilePath
        .FileName = FileSpec
        .Execute
'       Выход, если файлы не найдены
        If .FoundFiles.Count = 0 Then
            MsgBox "No files were found"
            Exit Sub
        End If
    End With
```

```
'   Просмотр и обработка файлов
    For i = 1 To FS.FoundFiles.Count
        Call ProcessFiles(FS.FoundFiles(i))
    Next i
End Sub
```

В этом примере используется три дополнительных файла, которые также можно найти на компакт-диске: `Text01.txt`, `Text02.txt` и `Text03.txt`. Вам необходимо будет изменить процедуру, чтобы она позволила импортировать другие текстовые файлы. Эта процедура использует объект `FileSearch`, поэтому она работает только в Excel версии 2000 и выше.

Соответствующие заданному критерию файлы получает объект `FileSearch`, а процедура использует для обработки файлов цикл `For-Next`. В цикле обработка выполняется процедурой `ProcessFiles`, показанной ниже. Эта простая процедура использует метод `OpenText` для импорта файла и вставки в него пяти формул. Конечно, вы можете заменить такую процедуру собственной, соответствующей более конкретной задаче.

```
Sub ProcessFiles(FileName As String)
'   Импорт файла
    Workbooks.OpenText FileName:=FileName, _
        Origin:=xlWindows, _
        StartRow:=1, _
        DataType:=xlFixedWidth, _
        FieldInfo:= _
        Array(Array(0, 1), Array(3, 1), Array(12, 1))
'   Ввод формул суммирования
    Range("D1").Value = "A"
    Range("D2").Value = "B"
    Range("D3").Value = "C"
    Range("E1:E3").Formula = "=COUNTIF(B:B,D1)"
    Range("F1:F3").Formula = "=SUMIF(B:B,D1,C:C)"
End Sub
```

Функции, полезные для использования в программах VBA

В данном разделе представлены некоторые "практичные" функции, которые будут использоваться в ваших собственных приложениях либо помогут в создании аналогичных функций. Эти функции наиболее полезны, когда они вызываются из другой процедуры VBA. Следовательно, они объявляются с ключевым словом `Private` и не отображаются в диалоговом окне Excel Мастер функций.

Примеры, приведенные в этом разделе, можно найти на прилагаемом к книге компакт-диске.

Функция FileExists

Данная функция получает один аргумент (путь и имя файла) и возвращает `ИСТИНА`, если файл существует.

```
Private Function FileExists(fname) As Boolean
'  Возвращает ИСТИНА, если файл существует
  FileExists = (Dir(fname) <> "")
End Function
```

Функция FileNameOnly

Функция получает один аргумент (путь и имя файла) и возвращает только имя файла. Другими словами, функция обрезает путь.

```
Private Function FileNameOnly(pname) As String
'   Возвращает имя файла из пути/имени файла
    Dim i As Integer, length As Integer, temp As String
    length = Len(pname)
    temp = ""
    For i = length To 1 Step -1
        If Mid(pname, i, 1) = Application.PathSeparator Then
            FileNameOnly = temp
            Exit Function
        End If
        temp = Mid(pname, i, 1) & temp
    Next i
    FileNameOnly = pname
End Function
```

Функция `FileNameOnly` выполняется для любого пути и имени файла (даже если файл не существует). Если файл существует, то следующая функция более просто удаляет путь и возвращает имя файла.

```
Private Function FileNameOnly2(pname) As String
  FileNameOnly2 = Dir(pname)
End Function
```

Функция PathExists

Функция получает один аргумент (путь) и возвращает `ИСТИНА`, если путь существует.

```
Private Function PathExists(pname) As Boolean
'   Возвращает ИСТИНА, если путь существует
    Dim x As String
    On Error Resume Next
    x = GetAttr(pname) And 0
    If Err = 0 Then PathExists = True _
      Else PathExists = False
End Function
```

Функция RangeNameExists

Функция получает один аргумент (название диапазона) и возвращает `ИСТИНА`, если в активной рабочей книге существует указанное название диапазона.

```
Private Function RangeNameExists(nname) As Boolean
'   Возвращает ИСТИНА, если название диапазона существует
    Dim n As Name
    RangeNameExists = False
    For Each n In ActiveWorkbook.Names
        If UCase(n.Name) = UCase(nname) Then
            RangeNameExists = True
            Exit Function
        End If
    Next n
End Function
```

Функция SheetExists

Функция получает один аргумент (название рабочего листа) и возвращает `ИСТИНА`, если данный рабочий лист существует в активной рабочей книге.

```
Private Function SheetExists(sname) As Boolean
'   Возвращает ИСТИНА, если лист существует в активной рабочей книге
    Dim x As Object
    On Error Resume Next
    Set x = ActiveWorkbook.Sheets(sname)
    If Err = 0 Then SheetExists = True _
        Else SheetExists = False
End Function
```

Функция WorkbookIsOpen

Функция получает один аргумент (название рабочей книги) и возвращает `ИСТИНА`, если данная рабочая книга открыта.

```
Private Function WorkbookIsOpen(wbname) As Boolean
'   Возвращает ИСТИНА, если рабочая книга открыта
    Dim x As Workbook
    On Error Resume Next
    Set x = Workbooks(wbname)
    If Err = 0 Then WorkbookIsOpen = True _
        Else WorkbookIsOpen = False
End Function
```

Проверка принадлежности к коллекции

Следующая функция представляет "групповую" функцию, которую можно использовать для определения, является ли объект членом коллекции.

```
Private Function IsInCollection(Coln As Object, _
  Item As String) As Boolean
    Dim Obj As Object
    On Error Resume Next
    Set Obj = Coln(Item)
    IsInCollection = Not Obj Is Nothing
End Function
```

Данная функция имеет два аргумента: коллекцию (объект) и элемент (строка), который может являться или не являться членом данной коллекции. Функция будет создавать переменную объекта, представляющую элемент в коллекции. Если попытка успешна, функция возвращает `True`; в противном случае функция возвращает `False`.

Вы можете использовать функцию `IsInCollection` вместо трех других функций, перечисленных в этой главе: `RangeNameExists`, `SheetExists` и `WorkbookIsOpen`. Чтобы определить, содержится ли в активной книге диапазон с названием `Data`, вызовите функцию `IsInCollection` со следующим оператором.

```
MsgBox IsInCollection(ActiveWorkbook.Names, "Data")
```

Для определения, открыта ли рабочая книга с названием `Budget`, используйте такой оператор.

```
MsgBox IsInCollection(Workbooks, "budget.xls")
```

С целью определить, содержит ли активная рабочая книга лист с названием `Лист1`, используйте оператор

```
MsgBox IsInCollection(ActiveWorkbook.Worksheets, "Лист1")
```

Получение значения из закрытой рабочей книги

В VBA не существует метода получения значения из закрытого файла рабочей книги. Однако вы можете воспользоваться возможностью управления ссылками на файлы, которая предоставляется в Excel. В настоящем разделе описана функция VBA (`GetValue`, показанная ниже), которая получает значение из закрытой книги. Эта задача выполняется в результате вызова макроса XLM.

```
Private Function GetValue(path, file, sheet, ref)
'   Получает значение из закрытой рабочей книги
    Dim arg As String

'   Проверка существования файла
    If Right(path, 1) <> "\" Then path = path & "\"
    If Dir(path & file) = "" Then
        GetValue = "Файл не найден"
        Exit Function
    End If

'   Создание аргумента
    arg = "'" & path & "[" & file & "]" & sheet & "'!" & _
      Range(ref).Range("A1").Address(, , xlR1C1)

'   Выполнение макроса XLM
    GetValue = ExecuteExcel4Macro(arg)
End Function
```

Функция `GetValue` имеет четыре аргумента:

- `path` — путь к закрытому файлу (например, `"d:\files"`);
- `file` — название рабочей книги (например, `"budget.xls"`);
- `sheet` — название рабочего листа (например, `"Лист1"`);
- `ref` — ссылка на ячейку (например, `"C4"`).

Следующая процедура демонстрирует, как используется функция `GetValue`. В этой процедуре отображается значение ячейки `A1` листа `Лист1` файла `99Budget.xls` (папка `XLFiles\Budget` на диске `C:`).

```
Sub TestGetValue()
  p = "C:\XLFiles\Budget"
  f = "99Budget.xls"
  s = "Лист1"
  a = "A1"
  MsgBox GetValue(p, f, s, a)
End Sub
```

Ниже приведен еще один пример. Эта процедура считывает 1200 значений (100 строк и 12 столбцов) из закрытого файла и помещает эти значения на активный рабочий лист.

```
Sub TestGetValue2()
    p = "c:\XLFiles\Budget"
    f = "99Budget.xls"
    s = "Sheet1"
    Application.ScreenUpdating = False
    For r = 1 To 100
        For c = 1 To 12
            a = Cells(r, c).Address
            Cells(r, c) = GetValue(p, f, s, a)
        Next c
```

```
    Next r
    Application.ScreenUpdating = True
End Sub
```

Функция GetValue не работает, если ее использовать в формуле рабочего листа. Эту функцию вообще не рекомендуется использовать в формуле. Вы можете просто создать формулу со ссылкой для получения значения из закрытого файла.

Полезные функции в формулах Excel

Примеры, приведенные в этом разделе, представляют пользовательские функции, которые можно использовать в формулах рабочего листа. Помните, что эти процедуры функций необходимо определить в модуле VBA (а не модуле кода соответствующей рабочей книги Эта книга (ThisWorkbook), листа или формы).

Примеры из этого раздела можно найти на прилагаемом к книге компакт-диске.

Получение информации о форматировании ячейки

Данный раздел содержит ряд пользовательских функций, возвращающих информацию о форматировании ячейки. Такие функции используются при сортировке данных на основе форматирования (например, в случае, когда ячейки, выделенные полужирным шрифтом, должны располагаться рядом).

Вскоре вы сможете убедиться, что эти специальные функции не всегда обновляются автоматически — изменение форматирования не запускает команду пересчета формул в Excel. Чтобы вызвать глобальный пересчет формул (и обновить все пользовательские функции), нажмите <Ctrl+Alt+F9>.

Следующая функция возвращает ИСТИНА, если аргумент, состоящий из одной ячейки, выделен полужирным шрифтом.

```
Function ISBOLD(cell) As Boolean
'  Возвращает ИСТИНА, если для ячейки задан полужирный шрифт
  ISBOLD = cell.Range("A1").Font.Bold
End Function
```

Следующая функция возвращает ИСТИНА, если ячейка (аргумент) выделена курсивом.

```
Function ISITALIC(cell) As Boolean
'  Возвращает ИСТИНА, если для ячейки задан курсив
  ISITALIC = cell.Range("A1").Font.Italic
End Function
```

Обе предыдущие функции возвращают ошибку, если ячейка имеет смешанное форматирование (например, полужирным шрифтом отображены только отдельные символы). Функция, приведенная ниже, возвращает ИСТИНА только тогда, когда все символы в ячейке выделены полужирным шрифтом.

```
Function ALLBOLD(cell) As Boolean
'  Возвращает ИСТИНА, если все символы в ячейке
'  выделены полужирным шрифтом
    If IsNull(cell.Font.Bold) Then
        ALLBOLD = False
    Else
```

```
        ALLBOLD = cell.Font.Bold
    End If
End Function
```

Функция `FILLCOLOR`, представленная далее, возвращает целое число, соответствующее индексу цвета фона ячейки (цвета заливки ячейки). Если ячейка не имеет заливки, то функция возвращает значение `4142`.

```
Function FILLCOLOR(cell) As Integer
'  Возвращает целое число, соответствующее
'  цвету фона ячейки
  FILLCOLOR = cell.Range("A1").Interior.ColorIndex
End Function
```

Отображение даты сохранения файла или вывода файла на печать

Рабочая книга Excel содержит несколько встроенных свойств документа, к которым можно получить доступ с помощью свойства `BuiltinDocumentProperties` объекта `Workbook`. Следующая функция возвращает дату и время последнего сохранения рабочей книги.

```
Function LASTSAVED()
    Application.Volatile
    LASTSAVED = ThisWorkbook. _
      BuiltinDocumentProperties("Last Save Time")
End Function
```

Показанная ниже функция напоминает предыдущую, но возвращает дату и время последнего вывода рабочей книги на печать или предварительного просмотра рабочей книги.

```
Function LASTPRINTED()
    Application.Volatile
    LASTPRINTED = ThisWorkbook. _
      BuiltinDocumentProperties("Last Print Date")
End Function
```

При использовании этих функций в формуле необходимо вызвать пересчет формул (комбинация клавиш <Ctrl+Alt+F9>), чтобы получить текущие значения данных свойств.

Существует еще несколько встроенных свойств, но Excel их не использует. Например, при попытке получить свойство, указывающее размер файла, будет выведено сообщение об ошибке.

Приведенные выше функции `LASTSAVE` и `LASTPRINTED` предназначались для сохранения в той рабочей книге, в которой они используются. В отдельных случаях требуется сохранить функцию в книге, отличной от той (например `personal.xls`), в которой она используется, или в надстройке. Поскольку все функции ссылаются на книгу `ЭтаКнига` (`ThisWorkbook`), то выполняться корректно не будут. Следуйте приведенным ниже инструкциям для создания универсальных функций. В приведенных процедурах используется метод `Application.Caller`, который возвращает объект `Range`. Этот объект указывает на ячейку, из которой вызывается функция. Взаимоотношения родительских и дочерних объектов детально рассмотрены далее в этой главе.

```
Function LastSaved2()
     Application.Volatile
     LastSaved2 = Application.Caller.Parent.Parent. _
          BuitinDocumentProperties("Last Save Time")
End Function

Function LASTPRINTED2()
     Application.Volatile
     LastSaved2 = Application.Caller.Parent.Parent. _
          BuitinDocumentProperties("Last Print Date")
End Function
```

Основы иерархии объектов

Как известно, объектная модель Excel представляет собой определенную структуру: объекты содержатся в других объектах. В верхней части этой иерархии находится объект Application. Excel содержит другие объекты, в которые, в свою очередь, вложены более низкоуровневые объекты и т.д. Следующая иерархия отображает, как в этой структуре представлен объект Range.

Объект Application

  Объект Workbook

    Объект Worksheet

      Объект Range

На языке объектно-ориентированного программирования родителем объекта Range (Диапазон) является объект Worksheet (Рабочий лист), в котором он содержится. Родителем объекта Worksheet является объект Workbook (Рабочая книга), содержащий этот рабочий лист, а родителем объекта Workbook является объект Application (приложение, т.е. Excel).

Как можно применить эту информацию на практике? Проанализируем функцию VBA SheetName, показанную ниже. Данная функция получает один аргумент (диапазон) и возвращает имя рабочего листа, который содержит указанный диапазон. При этом используется свойство Parent объекта Range. Свойство Parent возвращает объект, а именно объект, содержащий объект Range.

```
Function SheetName(ref) As String
  SheetName = ref.Parent.Name
End Function
```

Функция WorkbookName возвращает название рабочей книги для конкретной ячейки. Обратите внимание, что эта функция использует свойство Parent дважды. Первое свойство Parent возвращает объект Worksheet, а второе свойство Parent возвращает объект Workbook.

```
Function WorkbookName(ref) As String
  WorkbookName = ref.Parent.Parent.Name
End Function
```

Функция AppName, показанная далее, переносит это упражнение на следующий логический уровень, обращаясь к свойству Parent трижды. Такая функция возвращает имя объекта Application для заданной ячейки. Конечно, указанная функция всегда будет возвращать значение Microsoft Excel.

```
Function AppName(ref) As String
  AppName = ref.Parent.Parent.Parent.Name
End Function
```

Подсчет количества ячеек между двумя значениями

Следующая функция с названием COUNTBETWEEN возвращает количество значений в диапазоне (первый аргумент), которые попадают в область, заданную вторым и третьим аргументами.

```
Function COUNTBETWEEN(InRange, num1, num2) As Long
'   Подсчитывает количество значений между num1 и num2
    With Application.WorksheetFunction
        COUNTBETWEEN = .CountIf(InRange, ">=" & num1) - _
            .CountIf(InRange, ">" & num2)
    End With
End Function
```

Обратите внимание, что эта функция вызывает функцию Excel СЧЕТЕСЛИ (COUNTIF). По существу, функция COUNTBETWEEN является "оболочкой", которая может упростить формулы.

Ниже приведен пример формулы, использующей функцию COUNTBETWEEN. Формула возвращает количество ячеек в диапазоне A1:A100, больше или равных 10 и меньше или равных 20.

```
=COUNTBETWEEN(A1:A100;10;20)
```

Применяйте эту функцию VBA, чтобы не вводить следующую длинную формулу.

```
=(СЧЕТЕСЛИ(A1:A100; ">=10"))-СЧЕТЕСЛИ(A1:A100; ">=20"))
```

Подсчет количества видимых ячеек в диапазоне

Функция COUNTVISIBLE, представленная ниже, получает аргумент (диапазон) и возвращает количество видимых ячеек в этом диапазоне. Ячейка не отображается, если она находится в скрытой строке или скрытом столбце.

```
Function COUNTVISIBLE(rng)
'   Подсчитывает видимые ячейки
    Dim CellCount As Long
    Dim cell As Range
    Application.Volatile
    CellCount = 0
    Set rng = Intersect(rng.Parent.UsedRange, rng)
    For Each cell In rng
       If Not IsEmpty(cell) Then
          If Not cell.EntireRow.Hidden And _
             Not cell.EntireColumn.Hidden Then _
             CellCount = CellCount + 1
        End If
    Next cell
    COUNTVISIBLE = CellCount
End Function
```

Эта функция циклически просматривает все ячейки диапазона и сначала проверяет, пуста ли ячейка. Если ячейка не пустая, то функция проверят, скрыты ли строка и столбец ячейки. При условии, если либо строка, либо столбец скрыты, переменная CellCount увеличивает значение на 1.

Функция COUNTVISIBLE используется при работе с автофильтрами или сворачивании уровней структуры. В обоих этих случаях часто применяются скрытые строки.

СОВЕТ

Функция Excel ПРОМЕЖУТОЧНЫЕ.ИТОГИ (с первым аргументом 2 или 3) также может применяться для подсчета видимых ячеек в списке автофильтрации.

В Excel 2003 вы можете добавить 100 к аргументу функции ПРОМЕЖУТОЧНЫЕ.ИТОГИ. В результате функция ПРОМЕЖУТОЧНЫЕ.ИТОГИ будет обрабатывать только отображаемые на экране (не скрытые) ячейки. В предыдущих версиях программы функция ПРОМЕЖУТОЧНЫЕ.ИТОГИ учитывала все ячейки: как скрытые, так и отображаемые.

Определение последней непустой ячейки в столбце или строке

В данном разделе представлены две важные функции: LASTINCOLUMN — возвращает содержимое последней непустой ячейки в столбце, а также LASTINROW, которая возвращает содержимое последней непустой ячейки в строке. Каждая функция получает один аргумент — диапазон ячеек. Этот диапазон может представлять собой полный столбец (для функции LASTINCOLUMN) или полную строку (для функции LASTINROW). Если аргумент не является полным столбцом или строкой, функция использует столбец или строку левой верхней ячейки в диапазоне. Например, следующая формула возвращает последнее значение в столбце B.

```
=LASTINCOLUMN(B5)
```

Представленная далее формула возвращает последнее значение в строке 7.

```
=LASTINROW(C7:D9)
```

ФУНКЦИЯ LASTINCOLUMN

Функция LASTINCOLUMN показана ниже.

```
Function LASTINCOLUMN(rng As Range)
    Application.Volatile
    Set LastCell = rng.Parent.Cells(Rows.Count, rng.Column) _
       .End(xlUp)
    LASTINCOLUMN = LastCell.Value
    If IsEmpty(LastCell) Then LASTINCOLUMN = ""
    If rng.Parent.Cells(Rows.Count, rng.Column) <> "" Then _
        LASTINCOLUMN = rng.Parent.Cells(Rows.Count, rng.Column)
End Function
```

Эта функция довольно сложная, поэтому ниже приведено несколько замечаний, которые помогут вам в ней разобраться.

- Оператор Application.Volatile вызывает выполнение функции всякий раз, когда пересчитываются формулы на рабочем листе.
- Оператор Rows.Count возвращает количество строк на рабочем листе. Используется именно он, а не жестко заданное значение 65536, из соображений совместимости (новые версии Excel могут включать большее количество строк на рабочем листе).
- rng.Column возвращает номер столбца левой верхней ячейки в аргументе rng.
- Благодаря ссылке rng.Parent функция работает корректно, даже если аргумент rng ссылается на другой лист или рабочую книгу.
- Метод End (с аргументом xlUp) эквивалентен переходу к последней ячейке столбца и нажатию <End> и клавиши со стрелкой “вверх”.
- Функция IsEmpty проверяет, пуста ли ячейка. Если ячейка пуста, функция возвращает пустую строку. Без этого оператора пустой ячейке соответствовал бы результат 0.
- Последний оператор If проверяет последнюю ячейку в столбце. Если ячейка не пуста, функция возвращает содержимое этой ячейки.

Функция LASTINROW

Функция LASTINROW представлена ниже. Она во многом напоминает функцию LASTINCOLUMN.

```
Function LASTINROW(rng As Range)
    Application.Volatile
    Set LastCell = rng.Parent.Cells(rng.Row, Columns.Count) _
        .End(xlToLeft)
    LASTINROW = LastCell.Value
    If IsEmpty(LastCell) Then LASTINROW = ""
    If rng.Parent.Cells(rng.Row, Columns.Count) <> "" Then _
        LASTINROW = rng.Parent.Cells(rng.Row, Columns.Count)
End Function
```

Соответствует ли строка шаблону?

Функция ISLIKE довольно проста (и очень полезна). Она возвращает ИСТИНА, если строка соответствует заданному шаблону.

Функция ISLIKE показана далее. Как видно, она представляет собой "оболочку", позволяющую использовать в формулах мощный оператор VBA Like.

```
Function ISLIKE(text As String, pattern As String) As Boolean
'  Возвращает ИСТИНА, если первый аргумент такой, как второй
    If text Like pattern Then ISLIKE = True _
        Else ISLIKE = False
End Function
```

Функция ISLIKE имеет два аргумента:

- text — тестовая строка или ссылка на ячейку, содержащую текстовую строку;
- pattern — строка, содержащая групповые символы согласно следующему списку.

Символ(ы) в шаблоне	Соответствует в тексте
?	Любой отдельный символ
*	Ноль и больше символов
#	Любая отдельная цифра (0–9)
[список_символов]	Любой отдельный символ из список_символов
[!список_символов]	Любой отдельный символ, не принадлежащий список_символов

Представленная ниже формула возвращает ИСТИНА, так как * соответствует любому количеству символов. Она возвращает ИСТИНА, если первый аргумент — любой текст, начинающийся с g.

```
=ISLIKE("guitar"; "g*")
```

Следующая формула возвращает ИСТИНА, так как ? соответствует любому отдельному символу. Если бы первым аргументом функции был "Unit12", то функция возвращала бы ЛОЖЬ.

```
=ISLIKE("Unit1"; "Unit?")
```

Показанная далее формула возвращает ИСТИНА, так как первый аргумент является одним из символов списка во втором аргументе.

```
=ISLIKE("a"; "[aeiou]")
```

Следующая формула возвращает ИСТИНА, если ячейка A1 содержит один из символов: a, e, i, o, u, A, E, I, O, U. При использовании функции UPPER в аргументе функция становится нечувствительной к регистру.

```
=ISLIKE(UPPER(A1); UPPER("[aeiou]"))
```

Представленная далее формула возвращает ИСТИНА, если в ячейке A1 находится значение, начинающееся с 1 и состоящее ровно из трех цифр (т.е. любое целое число от 100 до 199).

```
=ISLIKE(A1; "1##")
```

Извлечение из строки n-го элемента

ExtractElement — специальная функция рабочего листа (которую можно также вызвать из процедуры VBA), которая помогает извлечь элемент из текстовой строки. Например, если ячейка содержит следующий текст, вы можете использовать функцию ExtractElement для извлечения любых подстрок между дефисами.

```
123-456-789-0133-8844
```

Представленная далее формула, например, возвращает 0133, т.е. четвертый элемент в строке. Дефис (-) используется в строке как разделитель.

```
=ExtractElement("123-456-789-0133-8844",4,"-")
```

Функция ExtractElement имеет три аргумента:

- Txt — текстовая строка, из которой извлекается подстрока. Это может быть символьная строка или ссылка на ячейку;
- n — целое число, представляющее номер извлекаемого элемента;
- Separator — отдельный символ, используемый как разделитель.

НА ЗАМЕТКУ

Если вы зададите в качестве символа-разделителя пробел, то несколько пробелов будут рассматриваться как один, что не всегда соответствует требованиям. Если n превышает количество элементов в строке, функция возвращает пустую строку.

Ниже приведен код VBA функции ExtractElement.

```
Function ExtractElement(Txt, n, Separator) As String
'  Возвращает n-й элемент текстовой строки, где
'  элементы разделены указанным символом-разделителем.
   Dim AllElements As Variant
   AllElements = Split(Txt, Separator)
   EXTRACTELEMENT = AllElements(n-1)
End Function
```

В этой процедуре используется VBA-функция Split, возвращающая массив констант, из которого состоит текстовая строка. Массив начинается с нулевого элемента (а не первого), поэтому текущий элемент имеет индекс n-1.

Следует заметить, что функция Split появилась в Excel 2000. Если вы пользуетесь более ранней версией программы, то воспользуйтесь следующей процедурой.

```
Function ExtractElement(Txt, n, Separator) As String
'  Возвращает n-й элемент текстовой строки, где
'  элементы разделены указанным символом-разделителем.

    Dim Txt1 As String, TempElement As String
```

```
    Dim ElementCount As Integer, i As Integer

    Txt1 = Txt
'   Если разделитель - пробел, убрать лишние пробелы
    If Separator = Chr(32) Then Txt1 = Application.Trim(Txt1)

'   Добавление разделителя в конец строки
    If Right(Txt1, Len(Txt1)) <> Separator Then _
        Txt1 = Txt1 & Separator

'   Инициализация
    ElementCount = 0
    TempElement = ""

'   Извлечение каждого элемента
    For i = 1 To Len(Txt1)
        If Mid(Txt1, i, 1) = Separator Then
            ElementCount = ElementCount + 1
            If ElementCount = n Then
'               n-й символ найден, выход
                ExtractElement = TempElement
                Exit Function
            Else
                TempElement = ""
            End If
        Else
            TempElement = TempElement & Mid(Txt1, i, 1)
        End If
    Next i
    ExtractElement=""
End Function
```

Множественная функция

В этом примере рассматривается прием, который направлен на то, чтобы одна функция рабочего листа работала, как несколько функций. Например, ниже показан код VBA для специальной функции с названием StatFunction. Эта функция имеет два аргумента: диапазон (rng) и операция (op). В зависимости от значения op функция возвращает значение, вычисленное с помощью одной из следующих функций Excel: СРЗНАЧ (AVERAGE), СЧЕТ (COUNT), МАКС (MAX), МЕДИАНА (MEDIAN), МИН (MIN), МОДА (MODE), СТАНДОТКЛОН (STDEV), СУММ (SUM) или ДИСП (VAR).

Например, вы можете использовать эту функцию на рабочем листе следующим образом.

```
=STATFUNCTION(B1:B24;A24)
```

Результат формулы зависит от содержимого ячейки A24, которое представлено строкой: СРЗНАЧ (AVERAGE), СЧЕТ (COUNT), МАКС (MAX) и т.д. Вы можете применить этот прием в других типах функций.

```
Function STATFUNCTION(rng, op)
    Select Case UCase(op)
        Case "SUM"
            STATFUNCTION = WorksheetFunction.Sum(rng)
        Case "AVERAGE"
            STATFUNCTION = WorksheetFunction.Average(rng)
        Case "MEDIAN"
            STATFUNCTION = WorksheetFunction.Median(rng)
        Case "MODE"
            STATFUNCTION = WorksheetFunction.Mode(rng)
        Case "COUNT"
            STATFUNCTION = WorksheetFunction.Count(rng)
```

```
        Case "MAX"
            STATFUNCTION = WorksheetFunction.Max(rng)
        Case "MIN"
            STATFUNCTION = WorksheetFunction.Min(rng)
        Case "VAR"
            STATFUNCTION = WorksheetFunction.Var(rng)
        Case "STDEV"
            STATFUNCTION = WorksheetFunction.StDev(rng)
        Case Else
            STATFUNCTION = CVErr(xlErrNA)
    End Select
End Function
```

Функция SHEETOFFSET

Вероятно, вы знаете, что в Excel ограничена поддержка "трехмерных рабочих книг". Например, если требуется сослаться на другой рабочий лист в книге, включите в формулу имя рабочего листа. Данная проблема будет оставаться незначительной до тех пор, пока вы не попытаетесь скопировать формулу из одного листа на другой. Скопированные формулы продолжают ссылаться на первоначальное имя рабочего листа, и ссылки на листы не изменяются, как это реализуется в реальной трехмерной рабочей книге.

Пример, рассмотренный в этом разделе, представляет функцию VBA (под названием `SHEETOFFSET`), которая позволяет обращаться к листам относительным способом. Например, вы можете сослаться на ячейку `A1` предыдущего рабочего листа с помощью такой формулы.

```
=SHEETOFFSET(-1;A1)
```

Первый аргумент представляет лист и может быть положительным, отрицательным или нулевым. Второй аргумент должен быть ссылкой на одну ячейку. Вы можете скопировать эту формулу на другие листы, и в скопированных формулах будет использована относительная ссылка.

Ниже приведен код VBA функции `SHEETOFFSET`.

```
Function SHEETOFFSET(Offset As Long, Optional Cell As Variant)
    Dim WksIndex As Long, WksNum As Long
    Dim wks As Worksheet
    Application.Volatile
    If IsMissing(Cell) Then Set Cell = Application.Caller
    WksNum = 1
    For Each wks In Application.Caller.Parent.Parent.Worksheets
        If Application.Caller.Parent.Name = wks.Name Then
            SHEETOFFSET = Worksheets(WksNum + Offset).Range(Cell(1).Address)
            Exit Function
        Else
            WksNum = WksNum + 1
        End If
    Next wks
End Function
```

Возвращение максимального значения всех рабочих листов

Если необходимо определить максимальное значение в ячейке `B1` в нескольких рабочих листах, то используется следующая формула.

```
=МАКС(Лист1:Лист4!B1)
```

Эта формула возвращает максимальное значение ячейки `B1` для листов `Лист1`, `Лист4` и всех листов между ними.

Что же произойдет, если добавить после листа `Лист4` новый лист (`Лист5`)? Формула не будет автоматически изменена, поэтому ее необходимо отредактировать, чтобы включить ссылку на новый лист.

```
=МАКС(Лист1:Лист5!B1)
```

Функция `MAXALLSHEETS`, показанная ниже, получает аргумент (одна ячейка) и возвращает максимальное значение в этой ячейке во всех рабочих листах данной книги. Например, следующая формула возвращает максимальное значение в ячейке `B1` для всех листов книги.

```
=MAXALLSHEETS(B1)
```

При добавлении нового листа редактировать формулу не обязательно.

```
Function MAXALLSHEETS(cell)
    Dim MaxVal As Double
    Dim Addr As String
    Dim Wksht As Object
    Application.Volatile
    Addr = cell.Range("A1").Address
    MaxVal = -9.9E+307
    For Each Wksht In cell.Parent.Parent.Worksheets
        If Wksht.Name = cell.Parent.Name And _
          Addr = Application.Caller.Address Then
        ' избежание циклической ссылки
        Else
            If IsNumeric(Wksht.Range(Addr)) Then
                If Wksht.Range(Addr) > MaxVal Then _
                  MaxVal = Wksht.Range(Addr).Value
            End If
        End If
    Next Wksht
    If MaxVal = -9.9E+307 Then MaxVal = 0
    MAXALLSHEETS = MaxVal
End Function
```

Оператор `For Each` использует для доступа к рабочей книге следующее выражение.

```
cell.Parent.Parent.Worksheets
```

Родителем ячейки является рабочий лист, родителем рабочего листа — рабочая книга. Следовательно, цикл `For Each` проходит по всем рабочим листам в книге. Первый оператор `If` внутри цикла выполняет проверку, содержит ли ячейка, которая проверяется в данный момент, функцию. Если это так, то ячейка игнорируется во избежание циклической ссылки.

> НА ЗАМЕТКУ
>
> Функцию можно легко изменить, чтобы приспособить к выполнению вычислений в нескольких рабочих листах — определения минимального, среднего значения, суммы и т.д.

Возвращение массива случайных целых чисел без повторов

Функция `RANDOMINTEGERS`, представленная в этом разделе, возвращает массив целых чисел без повторов. Она предназначена для применения в формуле массива в нескольких ячейках.

Данная формула введена в весь диапазон с помощью комбинации клавиш <Ctrl+Shift+Enter>. Она возвращает массив целых чисел без повторов, упорядоченных произвольным образом. Так как формулу содержат 40 ячеек, целые числа указываются в диапазоне от 1 до 40.

Ниже приведен код функции RANDOMINTEGERS.

```
Function RANDOMINTEGERS()
    Dim FuncRange As Range
    Dim V() As Variant, ValArray() As Variant
    Dim CellCount As Double
    Dim i As Integer, j As Integer
    Dim r As Integer, c As Integer
    Dim Temp1 As Variant, Temp2 As Variant
    Dim RCount As Integer, CCount As Integer
    Randomize

'   Создание объекта Range
    Set FuncRange = Application.Caller

'   Возвращение ошибки, если диапазон FuncRange слишком большой
    CellCount = FuncRange.Count
    If CellCount > 1000 Then
        RANDOMINTEGERS = CVErr(xlErrNA)
        Exit Function
    End If

'   Присваивание переменных
    RCount = FuncRange.Rows.Count
    CCount = FuncRange.Columns.Count
    ReDim V(1 To RCount, 1 To CCount)
    ReDim ValArray(1 To 2, 1 To CellCount)

'   Заполнение массива произвольными номерами
'   и последовательными целыми числами
    For i = 1 To CellCount
        ValArray(1, i) = Rnd
        ValArray(2, i) = i
    Next i

'   Сортировка массива ValArray по произвольным числам
    For i = 1 To CellCount
        For j = i + 1 To CellCount
            If ValArray(1, i) > ValArray(1, j) Then
                Temp1 = ValArray(1, j)
                Temp2 = ValArray(2, j)
                ValArray(1, j) = ValArray(1, i)
                ValArray(2, j) = ValArray(2, i)
                ValArray(1, i) = Temp1
                ValArray(2, i) = Temp2
            End If
        Next j
    Next i

'   Занесение произвольных значений в массив V
    i = 0
    For r = 1 To RCount
        For c = 1 To CCount
            i = i + 1
            V(r, c) = ValArray(2, i)
        Next c
    Next r
    RANDOMINTEGERS = V
End Function
```

Расположение значений диапазона в произвольном порядке

Функция RANGERANDOMIZE, представленная ниже, получает в качестве аргумента диапазон и возвращает массив, содержащий этот диапазон с произвольно переставленными значениями.

```
Function RANGERANDOMIZE(rng)
    Dim V() As Variant, ValArray() As Variant
    Dim CellCount As Double
    Dim i As Integer, j As Integer
    Dim r As Integer, c As Integer
    Dim Temp1 As Variant, Temp2 As Variant
    Dim RCount As Integer, CCount As Integer
    Randomize

'   Возвращает ошибку, если диапазон слишком большой
    CellCount = rng.Count
    If CellCount > 1000 Then
        RANGERANDOMIZE = CVErr(xlErrNA)
        Exit Function
    End If

'   Присвоение переменных
    RCount = rng.Rows.Count
    CCount = rng.Columns.Count
    ReDim V(1 To RCount, 1 To CCount)
    ReDim ValArray(1 To 2, 1 To CellCount)

'   Заполнение массива произвольными числами
'   и значениями из rng
    For i = 1 To CellCount
        ValArray(1, i) = Rnd
        ValArray(2, i) = rng(i)
    Next i

'   Сортировка массива ValArray по произвольным числам
    For i = 1 To CellCount
        For j = i + 1 To CellCount
            If ValArray(1, i) > ValArray(1, j) Then
                Temp1 = ValArray(1, j)
                Temp2 = ValArray(2, j)
                ValArray(1, j) = ValArray(1, i)
                ValArray(2, j) = ValArray(2, i)
                ValArray(1, i) = Temp1
                ValArray(2, i) = Temp2
            End If
        Next j
    Next i

'   Занесение произвольных значений в массив V
    i = 0
    For r = 1 To RCount
        For c = 1 To CCount
            i = i + 1
            V(r, c) = ValArray(2, i)
        Next c
    Next r
    RANGERANDOMIZE = V
End Function
```

Код функции RANGERANDOMIZE подобен коду функции RANDOMINTEGERS.

На рис. 11.12 показано применение функции RANGERANDOMIZE. В диапазон B2:B11 введена такая формула массива.

```
{=RANGERANDOMIZE(A2:A11)}
```

Эта формула возвращает содержимое диапазона A2:A11, но в произвольном порядке.



	A	B	C	D
1	*Исходный*	*Случайные числа*		
2	1	9		
3	2	8		
4	3	2		
5	4	10		
6	5	5		
7	6	3		
8	7	4		
9	8	6		
10	9	1		
11	10	7		
12				
13				
14				

Рис. 11.12. Функция RANGERANDOMIZE *возвращает содержимое диапазона в произвольном порядке*

Вызов функций Windows API

Одна из самых важных возможностей VBA — поддержка функций, которые хранятся в динамически подключаемых библиотеках (Dynamic Link Libraries — DDL). Примеры настоящего раздела демонстрируют самые популярные функции Windows API.

Используемые объявления функций API зависят от версии Excel. Если вы попытаете применить 32-битовую функцию API в 16-битовой Excel 5, то получите ошибку. Аналогичным образом, при использовании 16-битовой функции в 32-битовой версии Excel 95 и выше также будет получена ошибка. Примеры данного раздела рассчитаны на использование 32-битовой версии Excel.

Эти и другие вопросы совместимости подробно обсуждаются в главе 25.

Определение связей с файлами

В Windows многие типы файлов ассоциируются с конкретным приложением. Эта связь позволяет загрузить файл в соответствующее приложение; для этого дважды щелкните мышью на файле.

Функция GetExecutable вызывает функцию Windows API с целью получить полный путь к приложению, связанному с указанным файлом. Например, в системе находится ряд файлов с расширением .txt — вероятно, один такой файл с названием Readme.txt в данный момент расположен в папке Windows. Функцию GetExecutable можно применять для определения полного пути приложения (которое запускается при двойном щелчке на выбранном файле).

Функции Windows API должны объявляться вверху модуля VBA.

```
Private Declare Function FindExecutableA Lib "shell32.dll"_
(ByVal lpFile As String, ByVal lpDirectory As String,_
ByVal lpResult As String) As Long
Function GetExecutable(strFile As String) As String
    Dim strPath As String
    Dim intLen As Integer
    strPath = Space(255)
    intLen = FindExecutableA(strFile, "\", strPath)
    GetExecutable = Trim(strPath)
End Function
```

На рис. 11.13 показан результат вызова функции `GetExecutable` с аргументом `c:\windows\readme.txt`. В данном случае этому файлу соответствует программа `NOTEPAD.EXE`.

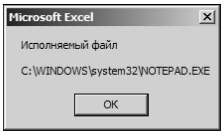


Рис. 11.13. Определение расположения приложения, соответствующего заданному файлу

Определение параметров принтера по умолчанию

В примере, приведенном в этом разделе, функция Windows API используется для получения информации об активном принтере. Данная информация содержится в одной текстовой строке. Программа разбирает эту строку и отображает информацию в более удобном для чтения формате.

```
Private Declare Function GetProfileStringA Lib "kernel32" _
  (ByVal lpAppName As String, ByVal lpKeyName As String, _
  ByVal lpDefault As String, ByVal lpReturnedString As _
  String, ByVal nSize As Long) As Long

Sub DefaultPrinterInfo()
    Dim strLPT As String * 255
    Dim Result As String
    Call GetProfileStringA _
      ("Windows", "Device", "", strLPT, 254)

    Result = Application.Trim(strLPT)
    ResultLength = Len(Result)

    Comma1 = Application.Find(",", Result, 1)
    Comma2 = Application.Find(",", Result, Comma1 + 1)

'   Получение названия принтера
    Printer = Left(Result, Comma1 - 1)

'   Получение драйвера
    Driver = Mid(Result, Comma1 + 1, Comma2 - Comma1 - 1)

'   Получение последней части записи об устройстве
    Port = Right(Result, ResultLength - Comma2)

'   Компоновка сообщения
    Msg = "Принтер:" & Chr(9) & Printer & Chr(13)
    Msg = Msg & "Драйвер:" & Driver & Chr(13)
    Msg = Msg & "Порт:" & Chr(9) & Port

'   Отображение сообщения
    MsgBox Msg, vbInformation, "Сведения о принтере по умолчанию"
End Sub
```

Свойство `ActivePrinter` объекта `Application` возвращает название активного принтера (и позволяет вам его изменить). Однако не существует прямого способа определить, какой драйвер принтера или порт используется. В этом “соль” данной функции.

На рис. 11.14 показано простое окно сообщения, полученное после выполнения этой процедуры.

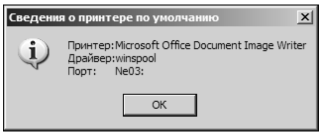


Рис. 11.14. Получение информации об активном принтере с помощью функции Windows API

Определение текущего видеорежима

В данном примере функции Windows API используются для определения текущего видеорежима системы. Если в приложении необходимо отобразить определенный объем информации на одном экране, то, зная размер экрана, можно, в соответствии с этим, правильно задать масштаб текста.

```
' Объявление 32-битовой API-функции
Declare Function GetSystemMetrics Lib "user32" _
  (ByVal nIndex As Long) As Long

Public Const SM_CXSCREEN = 0
Public Const SM_CYSCREEN = 1

Sub DisplayVideoInfo()
    vidWidth = GetSystemMetrics(SM_CXSCREEN)
    vidHeight = GetSystemMetrics(SM_CYSCREEN)

    Msg = "Текущий видеорежим: "
    Msg = Msg & vidWidth & " X " & vidHeight
    MsgBox Msg
End Sub
```

На рис. 11.15 показано окно сообщения, полученное при выполнении этой процедуры в системе с разрешением 1024×768.

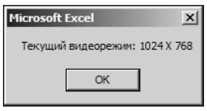


Рис. 11.15. Использование функции Windows API для определения разрешения

Добавление звука в приложение

Excel выполняет весьма ограниченное число действий со звуком. Самое эффективное средство — это команда VBA `Beep`. С помощью нескольких простых функций API ваше приложение сможет проигрывать файлы WAV или MIDI.

Не все системы поддерживают звук. Чтобы определить, поддерживает ли система звук, используется метод `CanPlaySounds`.

```
Sub CanPlaySound()
If Not Application.CanPlaySounds Then
  MsgBox "Извините, ваша система не воспроизводит звук"
End If
End Sub
```

ПРОИГРЫВАНИЕ ФАЙЛА WAV

Следующий пример содержит объявление функции API и простую процедуру проигрывания звукового файла с названием dogbark.wav (предполагается, что файл находится в той же папке, что и рабочая книга).

```
Private Declare Function PlaySound Lib "winmm.dll" _
  Alias "PlaySoundA" (ByVal lpszName As String, _
  ByVal hModule As Long, ByVal dwFlags As Long) As Long

Const SND_SYNC = &H0
Const SND_ASYNC = &H1
Const SND_FILENAME = &H20000

Sub PlayWAV()
    WAVFile = "dogbark.wav"
    WAVFile = ThisWorkbook.Path & "\" & WAVFile
    Call PlaySound(WAVFile, 0&, SND_ASYNC Or SND_FILENAME)
End Sub
```

В предыдущем примере файл WAV проигрывался несинхронно. Это означает, что выполнение продолжается, пока играет звук. Чтобы остановить выполнение программы на время проигрывания файла, используйте оператор

```
Call PlaySound(WAVFile, 0&, SND_SYNC Or SND_FILENAME)
```

ПРОИГРЫВАНИЕ ФАЙЛА MIDI

Если звуковой файл имеет формат MIDI, то необходимо применить вызов другой функции API. Файлы MIDI воспроизводит процедура PlayMIDI. Выполнение процедуры StopMIDI останавливает проигрывание файла MIDI. В данном примере применяется файл с названием xfiles.mid.

```
Private Declare Function mciExecute Lib "winmm.dll" _
  (ByVal lpstrCommand As String) As Long

Sub PlayMIDI()
    MIDIFile = "xfiles.mid"
    MIDIFile = ThisWorkbook.Path & "\" & MIDIFile
    mciExecute ("play " & MIDIFile)
End Sub

Sub StopMIDI()
    MIDIFile = "xfiles.mid"
    MIDIFile = ThisWorkbook.Path & "\" & MIDIFile
    mciExecute ("stop " & MIDIFile)
End Sub
```

ПРОИГРЫВАНИЕ ЗВУКА ИЗ ФУНКЦИИ РАБОЧЕГО ЛИСТА

Функция Alarm, показанная ниже, предназначена для применения в формуле рабочего листа. Она использует функцию Windows API для проигрывания звука, если ячейка соответствует определенному условию.

```
Declare Function PlaySound Lib "winmm.dll" _
  Alias "PlaySoundA" (ByVal lpszName As String, _
  ByVal hModule As Long, ByVal dwFlags As Long) As Long

Function ALARM(Cell, Condition)
    Dim WAVFile As String
    Const SND_ASYNC = &H1
    Const SND_FILENAME = &H20000
```

```
    If Evaluate(Cell.Value & Condition) Then
        WAVFile = ThisWorkbook.Path & "\sound.wav"
        Call PlaySound(WAVFile, 0&, SND_ASYNC Or SND_FILENAME)
        ALARM = True
    Else
        ALARM = False
    End If
End Function
```

Функция `Alarm` имеет два аргумента: ссылку на ячейку и “условие” (выраженное в виде строки). Например, следующая формула использует функцию `Alarm` для проигрывания файла WAV, если значение в ячейке `B13` больше или равно 1000.

```
=ALARM(B13; ">=1000")
```

Функция использует функцию VBA `Evaluate` для определения, соответствует ли значение ячейки заданному критерию. Если условие выполнено (и звук воспроизведен), функция возвращает `ИСТИНА`, в противном случае она возвращает `ЛОЖЬ`.

Чтение и запись параметров системного реестра

Многие приложения Windows используют системный реестр для хранения параметров (дополнительно о реестре рассказано в главе 4). Процедуры VBA могут считывать значения из реестра и записывать в него новые значения. Для этого необходимы следующие объявления функций Windows API.

```
Private Declare Function RegOpenKeyA Lib "ADVAPI32.DLL" _
    (ByVal hKey As Long, ByVal sSubKey As String, _
    ByRef hkeyResult As Long) As Long

Private Declare Function RegCloseKey Lib "ADVAPI32.DLL" _
    (ByVal hKey As Long) As Long

Private Declare Function RegSetValueExA Lib "ADVAPI32.DLL" _
    (ByVal hKey As Long, ByVal sValueName As String, _
    ByVal dwReserved As Long, ByVal dwType As Long, _
    ByVal sValue As String, ByVal dwSize As Long) As Long

Private Declare Function RegCreateKeyA Lib "ADVAPI32.DLL" _
    (ByVal hKey As Long, ByVal sSubKey As String, _
    ByRef hkeyResult As Long) As Long

Private Declare Function RegQueryValueExA Lib "ADVAPI32.DLL" _
    (ByVal hKey As Long, ByVal sValueName As String, _
    ByVal dwReserved As Long, ByRef lValueType As Long, _
    ByVal sValue As String, ByRef lResultLen As Long) As Long
```

Мною разработаны две функции-“оболочки”, упрощающие управление реестром: `GetRegistry` и `WriteRegistry`. Эти функции можно найти на прилагаемом к книге компакт-диске. Данная рабочая книга включает процедуру, демонстрирующую чтение и запись данных реестра.

ЧТЕНИЕ ДАННЫХ РЕЕСТРА

Функция GetRegistry возвращает раздел из указанного места регистра. Она располагает тремя аргументами.

- RootKey. Строка, представляющая ветвь реестра, к которой обращается функция. Данная строка может принимать одно из следующих значений.
  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_USER
  - HKEY_LOCAL_MACHINE
  - HKEY_USERS
  - HKEY_CURRENT_CONFIG
  - HKEY_DYN_DATA
- Path. Полный путь к разделу реестра, к которому обращается функция.
- RegEntry. Название параметра, который должна получить функция.

Если необходимо узнать текущий цвет строки заголовка активного окна, используйте функцию GetRegistry следующим образом (обратите внимание, что аргументы не чувствительны к регистру).

```
RootKey = "hkey_current_user"
Path = "Control Panel\Colors"
RegEntry = "ActiveTitle"
MsgBox GetRegistry(RootKey, Path, RegEntry), _
  vbInformation, Path & "\RegEntry"
```

В окне сообщения будет отображено три значения, представляющих цвета в модели RGB.

ЗАПИСЬ ДАННЫХ В РЕЕСТР

Функция WriteRegistry записывает значение в указанный раздел реестра. Если операция завершается успешно, функция возвращает ИСТИНА; в противном случае функция возвращает ЛОЖЬ. Функция WriteRegistry получает следующие аргументы (все они являются строками).

- RootKey. Строка, представляющая ветвь реестра, к которой обращается функция. Эта строка может принимать одно из перечисленных далее значений.
  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_USER
  - HKEY_LOCAL_MACHINE
  - HKEY_USERS
  - HKEY_CURRENT_CONFIG
  - HKEY_DYN_DATA
- Path. Полный путь в реестре. Если путь не существует, он будет создан.
- RegEntry. Название раздела реестра, в который записывается значение. Если раздел не существует, он добавляется в реестр.
- RegVal. Значение, которое записывается в реестр.

Ниже приведен пример, в котором записывается значение, представляющее время и дату запуска Excel. Эта информация сохраняется в разделе настроек Excel.

```
Sub Auto_Open()
    RootKey = "hkey_current_user"
    Path = "software\microsoft\office\10.0\excel\LastStarted"
    RegEntry = "DateTime"
    RegVal = Now()
    If WriteRegistry(RootKey, Path, RegEntry, RegVal) Then
        msg = RegVal & " сохранено в реестре"
          Else msg = "Произошла ошибка"
    End If
    MsgBox msg
End Sub
```

Если вы сохраните эту подпрограмму в личной книге макросов, то указанный параметр будет в дальнейшем автоматически обновляться при каждом запуске Excel.

Простой способ просмотра системного реестра

Если вам необходимо записать данные в системный реестр или извлечь их из реестра, то лучше прибегать к услугам функций `GetSetting` и `SaveSetting`.

В диалоговой справочной системе описаны эти две функции, которые детально не рассматриваются в этой главе. Они позволяют управлять данными следующего раздела реестра.

```
HKEY_CURRENT_USER\Software\VB and VBA Program Settings
```

Другими словами, с помощью этих функций вы можете управлять данными только одной ветви реестра, в которой сохраняются базовые настройки Excel.

Часть IV

Работа с пользовательскими формами

В этой части...

Глава 12

Создание собственных диалоговых окон

В ЭТОЙ ГЛАВЕ...

Диалоговые окна, возможно, являются самым важным элементом пользовательского интерфейса в программах Windows. Практически все программы в операционной системе Windows используют такие окна. Кроме того, многие пользователи знакомы с принципами управления диалоговыми окнами. Разработчики, создающие приложения Excel, могут программировать собственные диалоговые окна с помощью объектов `UserForm`. В этой главе рассмотрены следующие задачи.

- Использование окна ввода данных для получения информации от пользователя
- Использование окна сообщения для отображения сведений или эмуляции ответной реакции
- Выбор файла из диалогового окна
- Выбор папки
- Отображение диалоговых окон, встроенных в Excel

Перед тем, как приступить к изучению тонкостей создания диалоговых окон на основе пользовательских форм `UserForm`, вам стоит научиться использовать некоторые встроенные инструменты Excel, предназначенные для вывода диалоговых окон. Данная тема является основной в настоящей главе.

Перед созданием диалоговых окон…

В некоторых случаях можно избежать создания собственного диалогового окна. Воспользуйтесь одним из нескольких встроенных окон:

- окном ввода данных;
- окном сообщения;
- окном выбора файла;
- окном ввода имени файла и его расположения при сохранении;
- окном выбора папки;
- окном запроса данных.

Эти диалоговые окна будут рассматриваться в следующих разделах.

Использование окна ввода данных

Окно ввода данных — это простое диалоговое окно, которое позволяет пользователю ввести единственное значение. Например, можно применить окно ввода данных, чтобы предоставить пользователю возможность ввести текст, числа или даже диапазон значений. Для создания окна ввода используются две функции `InputBox`: одна в VBA, а вторая в Excel.

Функция InputBox в VBA

Данная функция имеет следующий синтаксис.

```
InputBox(Запрос[,Заголовок][,По_умолчанию][,xpos][,ypos][,Справка, Раздел])
```

- *Запрос* — указывает текст, отображаемый в окне ввода (обязательный параметр).
- *Заголовок* — определяет заголовок окна ввода (необязательный параметр).
- *По_умолчанию* — задает значение, которое отображается в окне ввода по умолчанию (необязательный параметр).
- *xpos, ypos* — определяют координаты верхнего левого угла окна ввода на экране (необязательные параметры).
- *Справка, Раздел* — указывают файл и раздел в справочной системе (необязательные параметры).

Функция `InputBox` запрашивает у пользователя единственное значение. Она всегда возвращает строку, поэтому вы должны будете преобразовать результат в числовое значение.

Текст, отображаемый в окне ввода, может достигать 1024 символов (длину допускается изменять в зависимости от ширины используемых символов). Кроме этого, вы можете указать заголовок диалогового окна, значение по умолчанию и координаты окна ввода на экране. Также в данном коде указывается раздел справочной системы со всеми вспомогательными сведениями. Если определить этот раздел, то в диалоговом окне будет отображена кнопка Справка.

В следующем примере, показанном на рис. 12.1, используется функция VBA `InputBox`, которая запрашивает у пользователя полное имя (имя и фамилию). Затем программа выделяет имя и отображает приветствие в окне сообщения.



Рис. 12.1. Результат выполнения функции VBA `InputBox`

```
Sub GetName()
    Dim UserName As String
    Dim FirstSpace As Integer
    Do Until UserName <> ""
        UserName = InputBox("Введите свое имя: ", _
            "Кто Вы?")
    Loop
    FirstSpace = InStr(UserName, " ")
    If FirstSpace <> 0 Then
        UserName = Left(UserName, FirstSpace - 1)
    End If
    MsgBox "Привет, " & UserName
End Sub
```

Этот пример доступен на прилагаемом к книге компакт-диске.

Обратите внимание: функция `InputBox` вызывается в цикле `Do Until`. Это позволяет убедиться в том, что данные введены в окно. Если пользователь щелкнет на кнопке Отмена (Cancel) или не введет текст, то переменная `UserName` будет содержать пустую строку, а окно ввода данных появится повторно. Далее в процедуре будет осуществлена попытка получить имя пользователя путем поиска первого символа пробела (для этого используется функция `InStr`). Таким образом, можно воспользоваться функцией `Left` для получения всех символов, расположенных слева от символа пробела. Если символ пробела не найден, то используется все введенное имя.

Как отмечалось ранее, функция `InputBox` всегда возвращает строку. Если строка, предоставленная в качестве результата выполнения функции `InputBox`, выглядит как число, ее можно преобразовать с помощью функции VBA `Val`. В противном случае следует применить функцию `InputBox` в Excel.

Метод InputBox в Excel

Использование метода Excel `InputBox` (вместо функции VBA `InputBox`) предоставляет три преимущества:

- можно задать тип возвращаемого значения;
- можно указать диапазон листа, обведя мышью ячейки листа;
- производится автоматическая проверка правильности введенных данных.

В данном случае метод `InputBox` имеет следующий синтаксис.

```
object.InputBox(Запрос,Заголовок,По_умолчанию,Слева,Сверху,Справка,Раздел,Тип)
```

- *Запрос* — указывает текст, отображаемый в окне ввода (обязательный параметр).
- *Заголовок* — определяет заголовок окна ввода (необязательный параметр).
- *По_умолчанию* — задает значение, которое отображается в окне ввода по умолчанию (необязательный параметр).
- *Слева, Сверху* – определяют координаты верхнего левого угла окна ввода на экране (необязательные параметры).
- *Справка, Раздел* — указывают файл и раздел в справочной системе (необязательные параметры).
- *Тип* — указывает код типа данных, который будет возвращаться методом (необязательный параметр). Его возможные значения перечислены в табл. 12.1.

Таблица 12.1. Коды типов, возвращаемых методом Excel *InputBox* данных

Код	Значение
0	Формула
1	Число
2	Строка (текст)
4	Булево значение (`ИСТИНА` или `ЛОЖЬ`)
8	Ссылка на ячейку как объект диапазона
16	Значение ошибки (такое как `#Н/Д`)
64	Массив значений

Метод Excel InputBox является достаточно гибким. Использование суммы приведенных выше значений позволяет возвратить несколько типов данных. Например, для отображения окна ввода, которое принимает текстовый или числовой тип данных, установите код в значение 3 (т.е. 1+2 или “число”+“текст”). Если в качестве кода типа данных применить значение 8, то пользователь сможет ввести в поле адрес ячейки или диапазона ячеек. Кроме того, пользователь имеет возможность указать диапазон на текущем рабочем листе.

Процедура EraseRange, которая приведена ниже, использует метод InputBox. Таким образом, пользователь может указать удаляемый диапазон (рис. 12.2). Адрес диапазона вводится вручную, мышь необходима для выделения диапазона на листе.

Метод InputBox с кодом 8 возвращает объект Range (обратите внимание на ключевое слово Set). После этого выбранный диапазон очищается (с помощью метода Clear). По умолчанию в поле окна ввода отображается адрес текущей выделенной ячейки. Если в окне ввода щелкнуть на кнопке Отмена (Cancel), то оператор On Error завершит процедуру.

```
Sub EraseRange()
    Dim UserRange As Range
    DefaultRange = Selection.Address
    On Error GoTo Canceled
    Set UserRange = Application.InputBox _
        (Prompt:="Удаляемый диапазон:", _
        Title:="Удаление диапазона", _
        Default:=DefaultRange, _
        Type:=8)
    UserRange.Clear
    UserRange.Select
Canceled:
End Sub
```

Этот пример доступен на прилагаемом к книге компакт-диске.

Еще одним преимуществом применения метода Excel InputBox является автоматическая проверка правильности введенных данных программой Excel. Если в примере GetRange ввести данные, не представляющие диапазон адресов, то Excel отобразит специальное сообщение и позволит пользователю повторить ввод данных (рис. 12.3).

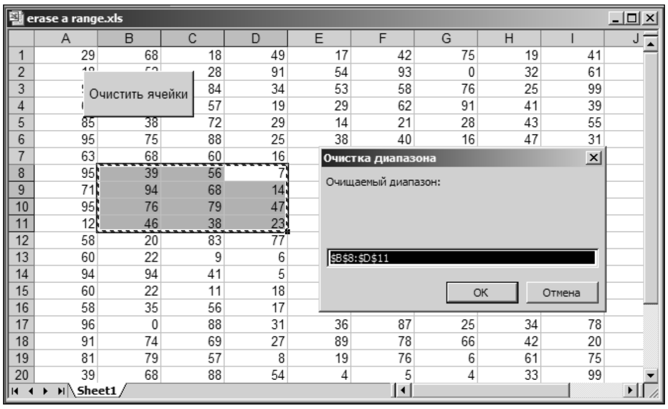


Рис. 12.2. Использование метода InputBox *для выделения и удаления диапазона*

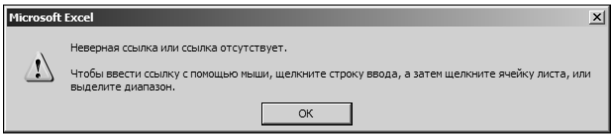


Рис. 12.3. Метод Excel `InputBox` *производит автоматическую проверку правильности введенных данных*

Функция VBA MsgBox

Функция VBA `MsgBox` представляет пользователю простой способ отображения сообщения. Также эта функция задает ответную реакцию пользователя на запрос (передает результат щелчка на кнопке OK или Отмена). Функция `MsgBox` применяется во многих примерах настоящей книги в качестве способа отображения значений переменных.

Ниже приведен синтаксис функции `MsgBox`.

```
MsgBox(Запрос[,Кнопки][,Заголовок][,Справка, Раздел])
```

- *`Запрос`* — определяет текст, который будет отображаться в окне сообщения (обязательный параметр).
- *`Кнопки`* — содержит числовое выражение, которые определяет кнопки, отображаемые в окне сообщения (необязательный параметр). Возможные значения приводятся в табл. 12.2.
- *`Заголовок`* — содержит заголовок окна сообщения (необязательный параметр).
- *`Справка, Раздел`* — указывают файл и раздел справочной системы (необязательные параметры).

Окна сообщений несложно изменить. Как правило, для этого применяется параметр *`Кнопки`*. (В табл. 12.2 приведен список констант, которые можно использовать в качестве значений этого параметра.) С его помощью указываются отображаемые кнопки, отмечается необходимость использования значка, определяется кнопка по умолчанию.

Таблица 12.2. Константы, используемые для выбора кнопок в функции *MsgBox*

Константа	Значение	Описание
`vbOKonly`	0	Отображает только кнопку OK
`vbOKCancel`	1	Отображает кнопки OK и Отмена
`vbAbortRetryIgnore`	2	Отображает кнопки Прервать, Повтор и Пропустить
`vbYesNoCancel`	3	Отображает кнопки Да, Нет и Отмена
`vbYesNo`	4	Отображает кнопки Да и Нет
`vbRetryCancel`	5	Отображает кнопки Повтор и Отмена
`vbCritical`	16	Отображает значок важного сообщения
`vbQuestion`	32	Отображает значок важного запроса
`vbExclamation`	48	Отображает значок предупреждающего сообщения
`vbInformation`	64	Отображает значок информационного сообщения
`vbDefaultButton1`	0	По умолчанию выделена первая кнопка
`vbDefaultButton2`	256	По умолчанию выделена вторая кнопка
`vbDefaultButton3`	512	По умолчанию выделена третья кнопка
`vbDefaultButton4`	768	По умолчанию выделена четвертая кнопка
`vbSystemModal`	4098	Все приложения приостанавливают свою работу до момента, пока пользователь ответит на запрос в окне сообщения (работает не во всех случаях)

Вы можете использовать функцию MsgBox в качестве процедуры (для отображения сообщения), а также присвоить возвращаемое этой функцией значение переменной. Функция MsgBox возвращает результат, который представляет кнопку, на которой щелкнул пользователь. Следующий пример отображает сообщение и не возвращает результат.

```
Sub MsgBoxDemo()
    MsgBox "Щелкните на кнопке OK для продолжения"
End Sub
```

Чтобы получить результат из окна сообщения, присвойте возвращаемое функцией MsgBox значение переменной. В следующем коде используется ряд встроенных констант (табл. 12.3), которые упрощают управление возвращаемыми функцией MsgBox значениями.

```
Sub GetAnswer()
    Ans = MsgBox("Продолжить?", vbYesNo)
    Select Case Ans
        Case vbYes
'       ...[код в случае Ans равно Yes]...
        Case vbNo
'       ...[код в случае Ans равно No]...
    End Select
End Sub
```

Таблица 12.3. Константы, возвращаемые функцией MsgBox

Константа	Значение	Нажатая кнопка
vbOK	1	OK
vbCancel	2	Отмена
vbAbort	3	Прервать
vbRetry	4	Повтор
vbIgnore	5	Пропустить
vbYes	6	Да
vbNo	7	Нет

Вам не обязательно использовать переменную для хранения результата выполнения функции MsgBox. Следующая процедура приводит к отображению окна сообщения с кнопками Да и Нет. Пока пользователь не щелкнет на кнопке Да, процедура будет продолжаться.

```
Sub GetAnswer2()
    If MsgBox("Продолжить?", vbYesNo) <> vbYes Then Exit Sub
'   ...[код для случая, когда на кнопке Да не щелкнули]...
End Sub
```

В следующем примере функции используется комбинация констант для отображения окна сообщения с кнопкой Да, кнопкой Нет и значком вопросительного знака. Вторая кнопка используется по умолчанию (рис. 12.4). Для простоты константы добавлены в переменную Config.

```
Private Function ContinueProcedure() As Boolean
    Dim Config As Integer
    Dim Ans As Integer
    Config = vbYesNo + vbQuestion + vbDefaultButton2
    Ans = MsgBox("Произошла ошибка. Продолжить?", Config)
    If Ans = vbYes Then ContinueProcedure = True _
        Else ContinueProcedure = False
End Function
```

Функция ContinueProcedure может вызываться из другой процедуры. Например, приведенный далее оператор вызывает функцию ContinueProcedure (которая отображает окно сообщения). Если функция возвращает значение ЛОЖЬ (т.е. пользователь щелкнул на кнопке Нет), то процедура будет завершена. В противном случае выполняется следующий оператор.

```
If Not ContinueProcedure Then Exit Sub
```

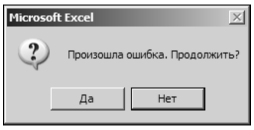


Рис. 12.4. Параметр Кнопки *функции* MsgBox *определяет кнопки, которые отображаются в окне сообщения*

Если в сообщении необходимо указать разрыв строки, воспользуйтесь константой vbCrLf (или vbNewLine) в необходимом месте. Отобразим сообщение в три строки.

```
Sub MultiLine()
    Dim Msg As String
    Msg = "Это первая строка" & vbCrLf
    Msg = Msg & "Вторая строка" & vbCrLf
    Msg = Msg & "Последняя строка"
    MsgBox Msg
End Sub
```

Вы также можете использовать в сообщении символ табуляции — для этого применяется константа vbTab. В приведенной далее процедуре окно сообщения используется для отображения диапазона значений размером 20×8 (рис. 12.5). В этом случае столбцы разделены с помощью константы vbTab. Новые строки вставляются с помощью константы vbCrLf. Функция MsgBox принимает в качестве параметра строку, длина которой не превышает 1023 символов. Такая длина задает ограничение на количество ячеек, которое можно отобразить в сообщении.

```
Sub ShowRange()
    Dim Msg As String
    Dim r As Integer, c As Integer
    Msg = ""
    For r = 1 To 20
        For c = 1 To 8
            Msg = Msg & Cells(r, c) & vbTab
        Next c
        Msg = Msg & vbCrLf
    Next r
    MsgBox Msg
End Sub
```

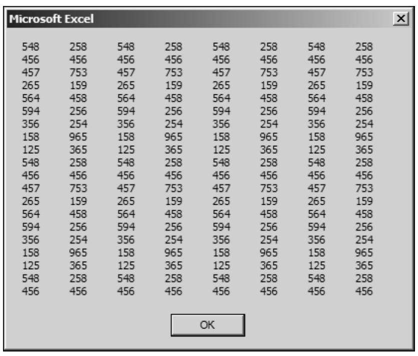


Рис. 12.5. Текст в этом окне сообщения содержит символы табуляции и разрыва строк

Глава 15 содержит пример кода VBA, который эмулирует поведение функции `MsgBox`.

Метод Excel GetOpenFilename

Если приложению необходимо получить от пользователя имя файла, то можно воспользоваться функцией `InputBox`, но этот подход часто приводит к возникновению ошибок. Более надежным считается использование метода `GetOpenFilename` объекта `Application`, который позволяет удостовериться, что приложение получило корректное имя файла (а также его полный путь).

Данный метод отображает стандартное диалоговое окно Открытие документа (которое появляется при выборе команды Файл⇨Открыть), но при этом указанный файл не открывается. Вместо этого метод возвращает строку, которая содержит путь и имя файла, выбранных пользователем. По окончании данного процесса с именем файла можно делать все, что угодно. Этот метод имеет следующий синтаксис (все параметры необязательные).

```
object.GetOpenFilename(Фильтр_файла, Индекс_фильтра, Заголовок, Подпись_кнопки,
Множественный_выбор)
```

- *Фильтр_файла* — содержит строку, определяющую критерий фильтрации файлов (необязательный параметр).
- *Индекс_фильтра* — указывает индексный номер того критерия фильтрации файлов, который используется по умолчанию (необязательный параметр).
- *Заголовок* — содержит заголовок диалогового окна (необязательный параметр). Если этот параметр не указать, то будет использован заголовок Открытие документа.
- *Подпись_кнопки* — применяется только в компьютерах Macintosh.
- *Множественный_выбор* — необязательный параметр. Если он имеет значение `ИСТИНА`, можно выбрать несколько имен файлов. По умолчанию данный параметр имеет значение `ЛОЖЬ`.

Аргумент *Фильтр_файла* определяет содержимое раскрывающегося списка Тип файлов. Аргумент состоит из строки, определяющей отображаемое в диалоговом окне значение, а также строки действительной спецификации типа файлов, в которой находятся групповые символы. Оба элемента аргумента разделены запятыми. Если этот аргумент не указать, то будет использовано значение по умолчанию.

```
" Все файлы (*.*),*.*"
```

Обратите внимание на первую часть строки `Все файлы (*.*)`. Это текст, отображаемый в раскрывающемся списке тип файлов. Вторая часть строки `*.*` указывает тип отображаемых файлов.

В следующих инструкциях переменной `Filt` присваивается строковое значение. Эта строка впоследствии используется в качестве аргумента *Фильтр_файла* метода `GetOpenFilename`. В данном случае диалоговое окно предоставит пользователю возможность выбрать один из четырех различных типов файлов (кроме варианта Все файлы). Если задать значение переменной `Filt`, то будет использоваться оператор конкатенации строки VBA. Этот способ упрощает управление громоздкими и сложными аргументами.

```
Filt = "Текстовые файлы (*.txt),*.txt," & _
       "Файлы печати(*.prn),*.prn," & _
       "Разделенные запятой(*.csv),*.csv," & _
       "ASCII (*.asc),*.asc," & _
       "Все файлы(*.*),*.*"
```

Аргумент *Индекс_фильтра* указывает значение аргумента *Фильтр_файла* по умолчанию. Аргумент *Заголовок* определяет текст, который отображается в заголовке окна. Если параметр *Множественный_выбор* имеет значение ИСТИНА, то пользователь может выбрать в окне несколько файлов. Имя каждого файла заносится в массив.

В следующем примере у пользователя запрашивается имя файла. При этом в поле типа файлов используется пять фильтров.

```
Sub GetImportFileName()
    Dim Filt As String
    Dim FilterIndex As Integer
    Dim FileName As Variant
    Dim Title As String

'   Настройка списка фильтров
    Filt = "Текстовые файлы (*.txt),*.txt," & _
           "Файлы печати (*.prn),*.prn," & _
           "Разделенные запятой (*.csv),*.csv," & _
           "ASCII (*.asc),*.asc," & _
           "Все файлы (*.*),*.*"

'   По умолчанию используется фильтр *.*
    FilterIndex = 5

'   Заголовок окна
    Title = "Выберите импортируемый файл"

'   Получение имени файла
    FileName = Application.GetOpenFilename _
        (FileFilter:=Filt, _
         FilterIndex:=FilterIndex, _
         Title:=Title)

'   При отмене выйти из окна
    If FileName = False Then
        MsgBox "Файл не выбран"
        Exit Sub
    End If

'   Отображение полного имени и пути
    MsgBox "Вы выбрали " & FileName
End Sub
```

На рис. 12.6 показано диалоговое окно, которое выводится на экран при выполнении этой процедуры.

Приведенный далее пример напоминает предыдущий. Разница заключается в том, что пользователь может, удерживая клавиши <Shift> или <Ctrl>, выбрать в окне несколько файлов. Обратите внимание, что событие использования кнопки Отмена определяется по наличию переменной массива `FileName`. Если пользователь не щелкнул на кнопке Отмена, то результирующий массив будет состоять как минимум из одного элемента. В этом примере список выбранных файлов отображается в окне сообщения.

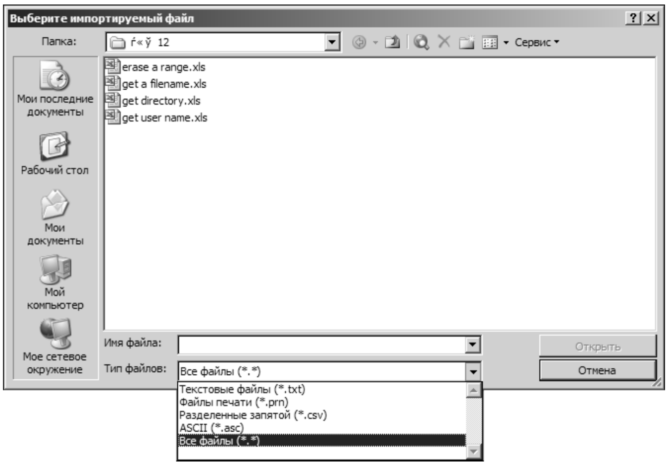


Рис. 12.6. Метод `GetOpenFilename` отображает пользовательское диалоговое окно

```
Sub GetImportFileName2()
    Dim Filt As String
    Dim FilterIndex As Integer
    Dim FileName As Variant
    Dim Title As String
    Dim i As Integer
    Dim Msg As String
'   Настройка фильтров файлов
    Filt  = "Текстовые файлы (*.txt),*.txt," & _
            "Файлы печати (*.prn),*.prn," & _
            "Разделенные запятой (*.csv),*.csv," & _
            "ASCII (*.asc),*.asc," & _
            "Все файлы (*.*),*.*"
'   По умолчанию используется фильтр *.*
    FilterIndex = 5

'   Заголовок окна
    Title = "Выберите импортируемый файл"

'   Get the file name
    FileName = Application.GetOpenFilename _
        (FileFilter:=Filt, _
         FilterIndex:=FilterIndex, _
         Title:=Title, _
         MultiSelect:=True)

'   Закрытие окна при использовании кнопки Отмена
    If Not IsArray(FileName) Then
        MsgBox "Файл не выбран"
        Exit Sub
    End If

'   Отображение полного имени и пути
    For i = LBound(FileName) To UBound(FileName)
        Msg = Msg & FileName(i) & vbCrLf
    Next i
    MsgBox "Вы выбрали:" & vbCrLf & Msg
End Sub
```

Обратите внимание, что переменная `FileName` определена как массив переменного типа (а не как строка в предыдущем примере). Причина заключается в том, что потенциально `FileName` может содержать массив значений, а не только одну строку.

Метод Excel GetSaveAsFilename

Данный метод имеет много общего с методом `GetOpenFilename`. Он отображает диалоговое окно Сохранение документа и позволяет пользователю выбрать (или указать) имя сохраняемого файла. В результате возвращается имя файла, но никакие действия не предпринимаются.

Этот метод имеет следующий синтаксис.

```
object.GetSaveAsFilename(Начальное_имя, Фильтр_файла, Индекс_фильтра, Заголовок,
Текст_кнопки)
```

- *Начальное_имя* — указывает предполагаемое имя файла (необязательный параметр).
- *Фильтр_файла* — содержит критерий фильтрации отображаемых в окне файлов (необязательный параметр).
- *Индекс_фильтра* — код критерия фильтрации файлов, который используется по умолчанию (необязательный параметр).
- *Заголовок* — определяет текст заголовка диалогового окна (необязательный параметр).
- *Текст_кнопки* — предназначен только для платформ Macintosh.

Получение имени папки

Если необходимо получить имя файла, то самым простым решением будет использование метода `GetOpenFilename` (как было показано выше). Но если необходимо получить имя папки, то ваши действия будут полностью зависеть от используемой версии Excel.

В настоящем разделе рассматривается два способа получения имени папки. Первый метод более сложен, но выполняется в Excel 97 и поздних версиях. Второй метод намного проще, но используется только в Excel 2002 и выше.

Использование функций Windows API для получения имени папки

В данном разделе рассматривается функция `GetDirectory`, которая отображает диалоговое окно, показанное на рис. 12.7. Эта функция возвращает строку, представляющую имя выбранной пользователем папки. Если пользователь щелкнет на кнопке Отмена, то функция возвратит пустую строку. Этот метод поддерживается в Excel 97 и более поздних версиях.

Функция `GetDirectory` принимает один необязательный строковый аргумент, который отображается в диалоговом окне. Если аргумент не указать, то вместо него диалоговое окно выведет сообщение Выберите папку.

НА КОМПАКТ-ДИСКЕ

На прилагаемом к книге компакт-диске содержится файл рабочей книги, демонстрирующей эту процедуру в действии.

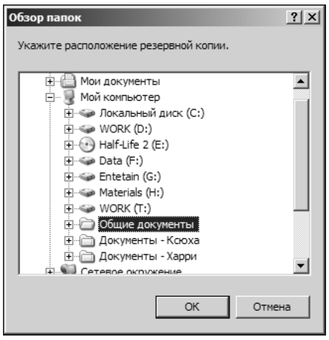


Рис. 12.7. Для отображения этого диалогового окна используется функция Windows API

Ниже представлен код объявления необходимых функций API, которые приведены в начале модуля рабочей книги. Данная функция использует собственный тип данных — BROWSEINFO.

```
'Объявление 32-х функций API
Declare Function SHGetPathFromIDList Lib "shell32.dll" _
  Alias "SHGetPathFromIDListA" (ByVal pidl As Long, ByVal _
  pszPath As String) As Long

Declare Function SHBrowseForFolder Lib "shell32.dll" _
  Alias "SHBrowseForFolderA" (lpBrowseInfo As BROWSEINFO) _
  As Long

Public Type BROWSEINFO
    hOwner As Long
    pidlRoot As Long
    pszDisplayName As String
    lpszTitle As String
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type
```

После объявления функции API вводится функция GetDirectory.

```
Function GetDirectory(Optional Msg) As String
    Dim bInfo As BROWSEINFO
    Dim path As String
    Dim r As Long, x As Long, pos As Integer

'   Корневая папка = Рабочий стол
    bInfo.pidlRoot = 0&

'   Заголовок окна
    If IsMissing(Msg) Then
        bInfo.lpszTitle = "Выберите папку"
    Else
        bInfo.lpszTitle = Msg
    End If

'   Тип возвращаемой папки
    bInfo.ulFlags = &H1

'   Отображение диалогового окна
    x = SHBrowseForFolder(bInfo)
```

```
'   Проверка результата
    path = Space$(512)
    r = SHGetPathFromIDList(ByVal x, ByVal path)
    If r Then
        pos = InStr(path, Chr$(0))
        GetDirectory = Left(path, pos - 1)
    Else
        GetDirectory = ""
    End If
End Function
```

Приведенная ниже простая процедура демонстрирует применение функции `GetDirectory` в коде VBA. Выполнение этой процедуры приводит к отображению диалогового окна. Как только пользователь щелкнет на кнопке OK, функция `MsgBox` отобразит полный путь к выбранной папке. Если пользователь щелкнет на кнопке Отмена, то появится окно с сообщением Отменено.

```
Sub GetAFolder1()
'   Для Excel 97 и более поздних версий
    Dim Msg As String
    Dim UserFile As String
    Msg = "Укажите расположение резервной копии"
    UserFile = GetDirectory(Msg)
    If UserFile = "" Then
        MsgBox "Отменено"
    Else
        MsgBox UserFile
    End If
End Sub
```

К сожалению, не существует простого способа указать начальную папку или папку по умолчанию.

Применение объекта FileDialog для выбора папки

Если все пользователи, на которых рассчитано приложение, используют Excel 2002 и выше, то рекомендуется прибегнуть к более простой методике, в которой применяется объект `FileDialog`.

Объект `FileDialog` впервые появился в Excel 2002. Таким образом, рассмотренная ниже методика не будет поддерживаться предыдущими версиями программы Excel.

Представленная ниже процедура отображает диалоговое окно, которое позволяет пользователю выбрать папку. Выбранная папка (или сообщение Отменено) отображается с помощью функции `MsgBox`.

```
Sub GetAFolder2()
'   Для Excel 2002
    With Application.FileDialog(msoFileDialogFolderPicker)
        .InitialFileName = Application.DefaultFilePath & "\"
        .Title = "Укажите расположение резервной копии"
        .Show
        If .SelectedItems.Count = 0 Then
            MsgBox "Отменено"
        Else
            MsgBox .SelectedItems(1)
        End If
    End With
End Sub
```

Объект `FileDialog` позволяет указать начальную папку. Для этого необходимо задать значение свойства `InitialFileName`. В нашем случае в качестве начальной папки по умолчанию используется папка сохранения файлов Excel.

Отображение встроенных диалоговых окон Excel

VBA может выполнять команды меню Excel. Если в процессе выполнения кода VBA на экране отображаются диалоговые окна, то в коде можно автоматически "выбирать опции" диалогового окна (хотя при этом само диалоговое окно и не отображается). Например, следующий оператор VBA аналогичен выбору команды Правка⇨Перейти, указанию диапазона `A1:C3` и щелчку на кнопке OK. Однако в данном случае диалоговое окно Переход не отображается (это именно то, что необходимо).

```
Application.Goto Reference:=Range("A1:C3")
```

В некоторых случаях вам может понадобится отобразить одно из встроенных диалоговых окон Excel, чтобы пользователь мог выбрать в нем необходимые параметры. Для этого воспользуйтесь одним из двух способов.

- Получите доступ к коллекции `Dialogs` объекта `Application`.
- Непосредственно выберите соответствующую команду меню.

Использование коллекции Dialogs

Коллекция `Dialogs` объекта `Application` состоит из 258 элементов, которые представляют большую часть встроенных диалоговых окон Excel. Каждому элементу соответствует предопределенная константа, которая позволяет указать, какое именно диалоговое окно необходимо отобразить. Например, диалоговое окно Excel Переход представлено константой `xlDialogFormulaGoto`.

Воспользуйтесь методом `Show` для фактического отображения диалогового окна. Ниже приведен пример, в котором отображается диалоговое окно Переход (рис. 12.8).

```
Application.Dialogs(xlDialogFormulaGoto).Show
```

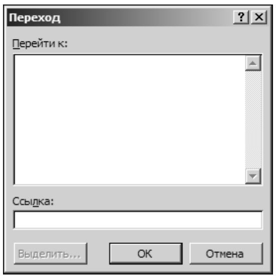


Рис. 12.8. Это диалоговое окно выведено оператором VBA

Когда на экране отображается диалоговое окно Переход, пользователь может указать именованный диапазон или адрес ячейки, к которой необходимо перейти. Указанное диалоговое окно появляется при выборе команды Правка⇨Перейти (также можно нажать клавишу <F5>).

Кроме того, вы вправе написать код, который будет определять, как пользователь завершил работу в диалоговом окне. Для этого воспользуйтесь переменной. В представленном ниже операторе переменная `Result` будет иметь значение `True`, если

пользователь щелкнет на кнопке OK, и `False`, если он щелкнет на кнопке Отмена или нажмет клавишу <Esc>.

```
Result = Application.Dialogs(xlDialogFormulaGoto).Show
```

Весьма странно, но переменная `Range` не содержит указанный в диалоговом окне Переход диапазон.

Следует отметить, что такое поведение не достаточно хорошо документировано. Диалоговые источники справочных сведений слишком краткие, они зачастую лишены информации о том, что отображение одного из встроенных в Excel диалоговых окон с помощью кода VBA не всегда приводит к тому же результату, что и выбор команды меню. Следовательно, вам ст*о*ит немного поэкспериментировать, чтобы удостовериться в правильности возвращаемого кодом результата.

При работе с диалоговым окном Переход вы сможете заметить, что при его вызове с помощью VBA кнопка Выделить остается неактивной. Данная кнопка обычно используется для отображения диалогового окна Выделение группы ячеек. Чтобы отобразить это диалоговое окно посредством кода VBA, воспользуйтесь таким оператором.

```
Application.Dialogs(xlDialogSelectSpecial).Show
```

Еще одной потенциальной проблемой является невозможность корректного отображения некоторых диалоговых окон с несколькими вкладками. Например, не существует способа отображения диалогового окна Формат ячеек одновременно со всеми вкладками. Вместо этого в VBA представлена возможность отображения каждой вкладки по отдельности. Представленный далее оператор отображает вкладку Выравнивание диалогового окна Формат ячеек (рис. 12.9).

```
Application.Dialogs(xlDialogAlignment).Show
```

Для того чтобы отобразить другие вкладки диалогового окна Формат ячеек, воспользуйтесь одной из констант: `xlDialogFormatNumber`, `xlDialogBorder`, `xlDialogCellProtection`, `xlDialogPatterns` или `xlDialogFontProperties`. Обратите внимание на отсутствие единого способа именования констант.

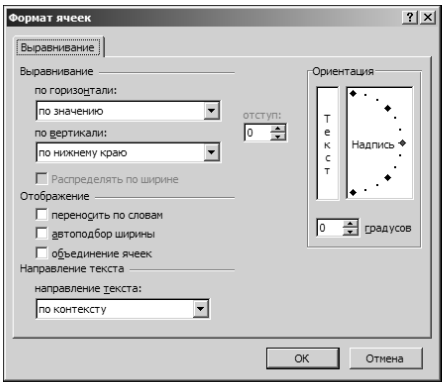


Рис. 12.9. Вкладка Выравнивание диалогового окна Формат ячеек

Получение дополнительной информации о встроенных диалоговых окнах

Список всех констант диалоговых окон можно получить в справочной системе или с помощью средства Object Browser. Следуйте приведенным ниже инструкциям для отображения членов коллекции `Dialogs` в окне Object Browser.

1. Активизировав модуль VBA, нажмите клавишу <F2> — будет отображено окно Object Browser.
2. В диалоговом окне Object Browser выберите в верхней части списка значение `Excel`.
3. Введите `xlDialog` во втором списке.
4. Щелкните на кнопке с изображением бинокля.

Попытка отобразить встроенное диалоговое окно в неверном контексте приведет к ошибке. Например, если выбрать последовательность на графике и попытаться отобразить диалоговое окно `xlDialogFontProperties`, то будет выдано сообщение об ошибке, так как это диалоговое окно не совместимо с таким типом выделения.

Использование аргументов во встроенных диалоговых окнах

Большая часть встроенных диалоговых окон поддерживает передачу аргументов, которые (обычно) соответствуют элементам управления диалогового окна. Например, диалоговое окно Формат ячеек (оно вызывается с помощью константы `xlDialogCellProtection`) принимает два аргумента: `locked` и `hidden`. Если необходимо отобразить диалоговое окно с уже установленными флажками, воспользуйтесь следующим оператором.

```
Application.Dialogs(xlDialogCellProtection).Show True, True
```

Аргументы для каждого из встроенных диалоговых окон перечисляются в справочной системе к программе VBE. Для того чтобы найти необходимый раздел в справочном руководстве, введите в качестве критерия поиска фразу **Built-In Dialog Box Argument List**. К сожалению, справочная система не содержит информации о назначении каждого из аргументов.

В соответствии с содержимым найденного раздела диалоговое окно Переход (которое вызывается с помощью константы `xlDialogFormulaGoTo`) принимает два аргумента: `reference` и `corner`. Аргумент `reference` используется по умолчанию для отображения диапазона в поле Ссылка. `Corner` — это логическое значение, которое определяет необходимость отображения выбранного диапазона таким образом, чтобы он находился в верхнем левом углу окна. Ниже приведен пример, в котором задействованы оба этих аргумента.

```
Application.Dialogs(xlDialogFormulaGoto). _
  Show Range("Z100"), True
```

Заметьте, что успешное освоение методов управления коллекцией `Dialogs` требует определенного времени, которое необходимо для проверки кода методом многочисленных проб и ошибок.

Непосредственный выбор команды меню

Еще один способ отображения встроенного диалогового окна требует наличия определенных знаний о панелях инструментов (которые официально называются объектами `CommandBar`). На данный момент достаточно знать, что команду меню можно выбрать программным образом. Это позволяет отображать диалоговое окно с помощью команды меню.

> Объекты `CommandBar` подробно рассматриваются в главах 22–23.

Следующий оператор аналогичен выбору команды Перейти в меню Правка.

```
Application.CommandBars("Worksheet Menu Bar"). _
   Controls("Правка").Controls("Перейти...").Execute
```

Если выполнить этот оператор, будет отображено диалоговое окно Переход. Обратите внимание, что все названия опций должны полностью совпадать (включая троеточие после `Перейти`).

В отличие от применения коллекции `Dialogs`, эта методика не позволяет активизировать элементы управления диалогового окна по умолчанию.

> В приведенном выше примере содержатся специфические для русской версии Excel ссылки на объект `CommandBar`. Следовательно, эти операторы будут работать только в русскоязычной версии Excel. В приложениях, разрабатываемых для других языковых версий Excel, можно воспользоваться методом `FindControl`, а также свойством `Id` команды меню. Дополнительная информация по этому вопросу приведена в главе 22.

В предыдущем разделе был рассмотрен вопрос доступа к коллекции `Dialogs`. В результате мы пришли к выводу, что не существует возможности вывода диалогового окна с несколькими вкладками. Эта проблема будет решена при выводе диалоговых окон в результате программного выбора команд меню. Следующий оператор отображает диалоговое окно Формат ячеек, содержащее все вкладки.

```
Application.CommandBars("Worksheet Menu Bar"). _
   Controls("Формат").Controls("Ячейки...").Execute
```

Кстати, метод `Execute` поддерживается и элементами управления на панелях инструментов, которые не отображают диалоговые окна. Но подобной возможности сложно придумать достойное применение. Целесообразнее обратиться к свойствам выделенных ячеек, воспользовавшись следующим оператором (изменяет начертание на полужирное).

```
Selection.Font.Bold = Not Selection.Font.Bold
```

Глава 13

Использование пользовательских форм

В ЭТОЙ ГЛАВЕ...

Разработчики приложений Excel всегда имели возможность создавать собственные диалоговые окна. Начиная с Excel 97, все заметно изменилось. Пользовательские диалоговые окна заменили неуклюжие диалоговые листы, и у разработчиков появилось намного больше возможностей по управлению собственными диалоговыми окнами. Но в целях совместимости Excel 97 и более поздние версии все еще поддерживают старые диалоговые листы Excel 5/95. Хорошей новостью является то, что формами `UserForm` управлять намного проще, кроме того, они предоставляют широкий набор новых возможностей.

- Создание, отображение и выгрузка пользовательских диалоговых окон.
- Описание элементов управления пользовательских диалоговых окон.
- Определение свойств элементов управления пользовательских диалоговых окон.
- Управление пользовательскими диалоговыми окнами с помощью процедур VBA.
- Пример создания пользовательского диалогового окна.
- Введение в типы событий, обрабатываемых пользовательскими диалоговым окнами и элементам управления.
- Настройка окна Toolbox.
- Список инструкций по созданию пользовательских диалоговых окон.

Excel позволяет относительно просто создавать собственные диалоговые окна в разрабатываемых приложениях. На самом деле с их помощью можно повторить внешний вид и поведение практически всех стандартных диалоговых окон Excel. Настоящая глава предлагает обзор возможностей пользовательских диалоговых окон.

Как Excel обрабатывает пользовательские диалоговые окна

Пользовательские диалоговые окна создаются на основе технологии пользовательских форм `UserForm`, к которым можно получить доступ из редактора Visual Basic.

Ниже приведена стандартная последовательность действий, которой следует придерживаться при создании пользовательского диалогового окна.

1. Вставьте новую форму `UserForm` в проект VBAProject рабочей книги.
2. Создайте процедуру, которая будет отображать форму `UserForm`. Эта процедура располагается в модуле кода VBA (а не в модуле кода формы `UserForm`).

3. Добавьте элементы управления в форму `UserForm`.
4. Настройте свойства добавленных элементов управления.
5. Создайте процедуры "обработчики событий" для элементов управления. Эти процедуры добавляются в модуль кода `UserForm` и выполняются при возникновении различных событий (например, при щелчке на кнопке).

Вставка новой формы UserForm

Для того чтобы добавить в проект форму `UserForm`, запустите VBE (нажмите <Alt+F11>, укажите рабочую книгу в окне проекта и выберите команду Insert⇨UserForm). Формы `UserForm` получат такие имена, как `UserForm1`, `UserForm2` и т.д.

СОВЕТ Чтобы упростить идентификацию, можно изменить имя формы `UserForm`. Выберите форму и воспользуйтесь окном Properties для изменения свойства `Name` (нажмите клавишу <F4>, если окно Properties не отображается на экране). На рис. 13.1 приведен пример окна Properties для новой пустой формы `UserForm`.

Рабочая книга может содержать любое количество пользовательских диалоговых окон. При этом каждая форма `UserForm` соответствует лишь одному пользовательскому диалоговому окну.

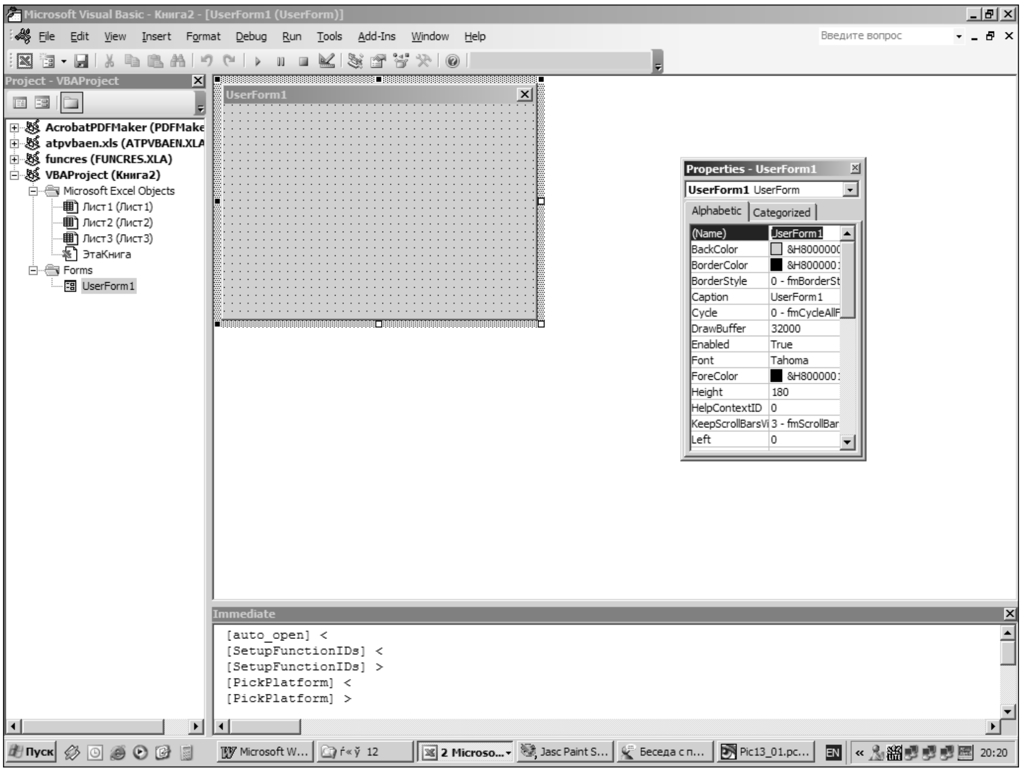


Рис. 13.1. Окно Properties *для пустого пользовательского диалогового окна*

Добавление элементов управления в пользовательское диалоговое окно

Чтобы добавить элементы управления в форму `UserForm`, воспользуйтесь окном Toolbox (в VBE отсутствуют команды меню, предназначенные для добавления элементов управления). Если окно Toolbox не отображено на экране, выберите команду View⇨Toolbox. Окно Toolbox показано на рис. 13.2.



Рис. 13.2. Воспользуйтесь окном Toolbox *для добавления элементов управления в пользовательское диалоговое окно*

Щелкните на той кнопке в окне Toolbox, которая соответствует добавляемому элементу управления. После этого щелкните внутри диалогового окна для создания элемента управления (используется размер элемента по умолчанию). Также можно щелкнуть на элементе управления и, перетащив его границы в диалоговом окне, задать необходимый размер в пользовательском диалоговом окне.

После добавления нового элемента управления ему назначается имя, которое состоит из названия типа элемента управления и числового кода. Например, если добавить элемент управления `CommandButton` в пустую форму `UserForm`, то этот элемент управления будет называться `CommandButton1`. Если добавить в окно второй элемент управления `CommandButton`, то он будет называться `CommandButton2`.

СОВЕТ

Рекомендуем переименовывать все элементы управления, которые управляются с помощью VBA-кода. Это позволит использовать более описательные имена объектов (например, `ProductListBox`), а не общие последовательно нумерованные названия (подобные `ListBox1`). Для того чтобы изменить имя элемента управления, воспользуйтесь окном Properties в VBE. Достаточно выделить необходимый объект и ввести новое имя.

Доступные элементы управления

В следующих разделах кратко описаны элементы управления, доступные в окне Toolbox.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Форма `UserForm` может также содержать другие элементы управления ActiveX. За дополнительной информацией обратитесь к разделу “Настройка панели инструментов Toolbox” далее в этой главе.

CheckBox

Элемент управления `CheckBox` предоставляет пользователю возможность выбрать один из двух вариантов: да или нет, истина или ложь, включить или выключить и т.д. Если элемент управления `CheckBox` установлен, то он имеет значение `True`, в противном случае значение равно `False`.

ComboBox

Элемент управления `ComboBox` подобен объекту `ListBox`. Отличие заключается в том, что `ComboBox` представляет раскрывающийся список, в котором в определенный момент времени отображается только одно значение. Кроме того, пользователю в поле списка разрешено вводить значение, которое не обязательно представляет одну из опций объекта `ComboBox`.

CommandButton

Каждое создаваемое диалоговое окно будет иметь как минимум один элемент управления `CommandButton`. Обычно используются объекты `CommandButton`, представляющие кнопки OK и Отмена.

Frame

Элемент управления `Frame` применяется в качестве оболочки для других элементов управления. Он добавляется в диалоговое окно либо в целях эстетики, либо из соображений логического группирования однотипных элементов управления. Элемент управления `Frame` потребуется вам в случае, когда диалоговое окно содержит более одного набора элементов управления `OptionButton`.

Image

Элемент управления `Image` используется для представления графического изображения, которое сохранено в отдельном файле или вставляется из буфера обмена. Кроме того, элемент управления `Image` незаменим при отображении в диалоговом окне логотипа компании. Графическое изображение сохраняется вместе с рабочей книгой. Таким образом, при передаче рабочей книги другому пользователю передавать вместе с ней копию графического файла не обязательно.

> **ВНИМАНИЕ**
> Отдельные изображения занимают много места на диске, что может привести к значительному увеличению размера рабочей книги. Чтобы получить наилучший результат, избегайте использования графических изображений и старайтесь вставлять графические файлы небольшого размера.

Label

Элемент управления `Label` отображает текст в диалоговом окне.

ListBox

Элемент управления `ListBox` предоставляет список опций, из которого пользователь может выбрать один вариант (или несколько). Элемент управления `ListBox` невероятно гибок в использовании. Например, вы вправе указать диапазон на листе, который содержит элементы списка. Этот диапазон может состоять из нескольких столбцов. Кроме того, элемент управления `ListBox` заполняется опциями также с помощью кода VBA.

MultiPage

Элемент управления `MultiPage` позволяет создавать диалоговые окна с несколькими вкладками, которые подобны появляющимся при выборе команды Сервис⇨Параметры. По умолчанию элемент управления `MultiPage` состоит из двух вкладок. Чтобы создать дополнительные вкладки, щелкните правой кнопкой мыши на существующей вкладке и выберите New Page из появившегося на экране контекстного меню.

OptionButton

Элемент управления OptionButton применяется при выборе пользователем одного варианта из нескольких. Эти элементы управления всегда группируются в диалоговом окне в наборы, содержащие не менее двух опций. Когда один элемент управления OptionButton выбран, все остальные элементы управления OptionButton текущей группы автоматически становятся неактивными.

Если пользовательское диалоговое окно содержит более одного набора элементов управления OptionButton, то каждый из таких наборов должен иметь собственное значение свойства GroupName. В противном случае все элементы управления OptionButton в диалоговом окне рассматриваются как члены одной группы. Также можно вставить элементы управления OptionButton в объект Frame, что приведет к их автоматическому группированию в текущем разделе.

RefEdit

Элемент управления RefEdit используется в том случае, когда пользователь должен выделить диапазон ячеек на листе.

ScrollBar

Элемент управления ScrollBar в некотором смысле подобен элементу управления SpinButton. Разница заключается в том, что пользователь может перетаскивать бегунок объекта ScrollBar для изменения значения с большим приращением. Элемент управления ScrollBar рекомендуется использовать при выборе значения из большого диапазона.

SpinButton

Элемент управления SpinButton позволяет выбрать значение в результате щелчка на одной из двух кнопок со стрелками. Одна из них применяется для увеличения значения, а вторая — для уменьшения. Элемент управления SpinButton часто используется совместно с элементами управления TextBox или Label, которые содержат текущее значение элемента управления SpinButton.

TabStrip

Элемент управления TabStrip подобен элементу управления MultiPage, однако использовать его сложнее. Элемент управления TabStrip, в отличие от MultiPage, не выступает контейнером для других объектов. Как правило, элемент управления MultiPage обладает более широкими возможностями.

TextBox

Элемент управления TextBox позволяет пользователям вводить в диалоговом окне текст.

ToggleButton

Элемент управления ToggleButton имеет два состояния: включен или выключен. Щелчок на кнопке приводит к изменению состояния на противоположное и к изменению внешнего вида кнопки. Этот элемент управления может иметь значение True (активен) или False (неактивен). Он не относится к “стандартным” элементам управления, потому использование двух элементов управления OptionButton или одного CheckBox может оказаться более удачным вариантом.

Настройка элементов управления пользовательского диалогового окна

После того, как элемент управления будет помещен в диалоговое окно, его можно переместить, а также изменить его размер. Воспользуйтесь стандартными методиками управления графическими объектами мышью.

> Существует возможность выделить несколько элементов управления. При этом удерживайте нажатой клавишу <Shift> и щелкайте на объектах. Можно также обвести указателем мыши все необходимые элементы управления.

Форма `UserForm` содержит вертикальные и горизонтальные направляющие, которые помогают выровнять добавленные в диалоговое окно элементы управления. При добавлении или перемещении элемент управления привязывается к направляющим, что облегчает упорядочивание таких элементов в окне. Если вы не используете направляющие, то можете отключить их, выбрав в VBE команду Tools⇨Options. В диалоговом окне Options перейдите на вкладку General и выберите соответствующие опции в разделе Form Grid Settings.

Меню Format окна VBE предоставляет несколько команд, которые позволяют точно разместить и выровнять элементы управления в диалоговом окне. Перед использованием этих команд необходимо указать элементы управления, к которым они будут применяться. Эти команды выполняют свои задачи так, как и ожидается. На рис. 13.3 показано диалоговое окно с несколькими элементами управления `OptionButton` в процессе выравнивания.

Использование элементов управления на рабочем листе

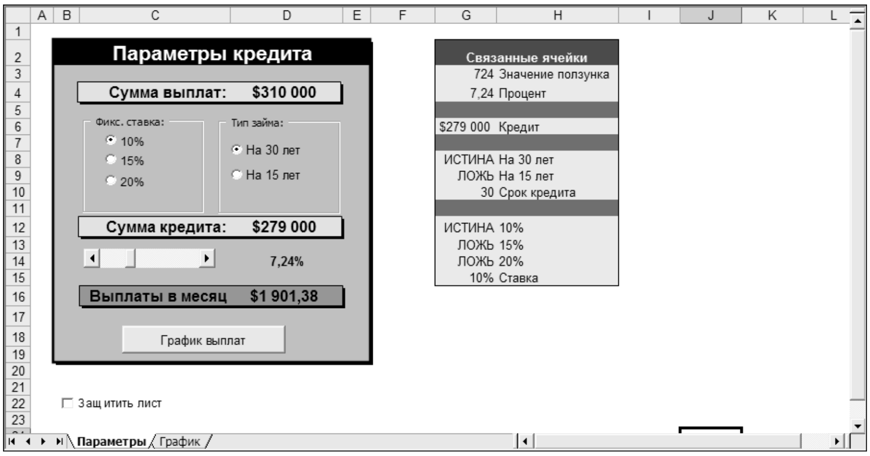
Элементы управления пользовательского диалогового окна могут встраиваться непосредственно на рабочий лист. Доступ к ним можно получить на панели инструментов Элементы управления (в Excel, а не в VBE). Добавление элементов управления на лист требует намного меньше усилий, чем создание пользовательского диалогового окна. Более того, в данном случае необязательно создавать макросы, так как элемент управления можно связать с любой ячейкой листа. Например, если на лист вставить элемент управления Флажок, то можно связать его с определенной ячейкой, задав значение его свойства `LinkedCell`. Когда элемент управления Флажок установлен, связанная с ним ячейка содержит значение `ИСТИНА`. Как только элемент управления сбрасывается, связанная с ним ячейка приобретает значение `ЛОЖЬ`.

Приведенный ниже рисунок представляет рабочий лист, на котором присутствуют встроенные элементы управления.

Добавление элементов управления на лист может оказаться непростой операцией, поскольку они вставляются из двух панелей инструментов.

- *Панель инструментов Формы*. Это внедряемые элементы управления, совместимые с Excel 5 и Excel 95.
- *Панель инструментов Элементы управления*. Это элементы управления ActiveX. Они являются подмножеством тех элементов управления, которые доступны в пользовательских диалоговых окнах. Такие элементы управления используются только в Excel 97 и в более поздних версиях, они не поддерживаются в Excel 5 и Excel 95.

Вы вправе использовать элементы управления на любой из этих двух панелей инструментов, но всегда помните об их различиях. Элементы управления панели инструментов Формы выполняют свои задачи иным образом, чем это делают элементы управления ActiveX.



При использовании панели инструментов Элементы управления для добавления элемента управления на лист Excel переходит в режим конструктора. В этом режиме можно изменять свойства любого элемента управления на листе, добавлять или редактировать процедуры обработки событий элемента управления или изменять размер и расположение элемента управления. Для того чтобы отобразить окно свойств для элемента управления ActiveX, щелкните правой кнопкой мыши на элементе управления и в появившемся контекстном меню выберите Свойства.

Для добавления простых кнопок можно использовать элемент управления Кнопка на панели инструментов Формы, так как он обеспечивает запуск макроса. Если же применить элемент управления Кнопка панели инструментов Элементы управления, то щелчок на этой кнопке приведет к запуску процедуры обработки события (например, `CommandButton1_Click`) в модуле кода для объекта `Лист`. Напрямую связать макрос с этим элементом управления нельзя.

Когда Excel находится в режиме конструктора, элементы управления использовать нельзя. Для того чтобы проверить работоспособность элементов управления, необходимо выйти из режима конструктора (щелкнув на кнопке Выход из режима конструктора на панели инструментов Элементы управления).

Эта и другие рабочие книги, которые демонстрируют применение элементов управления на рабочем листе, представлены на прилагаемом к книге компакт-диске.

СОВЕТ

При выделении нескольких элементов управления последний из выделенных имеет на рамке белые маркеры, а не черные, как все остальные. Элемент управления с белыми маркерами используется в качестве модели, по которой определяются размеры и расположение всех остальных элементов управления.

Изменение свойств элементов управления

Каждый элемент управления характеризуется набором параметров, которые определяют внешний вид и поведение элемента управления. Свойства элемента управления можно изменять в следующих случаях.

- В момент проектирования при разработке пользовательского диалогового окна. Для этого используется окно Properties.
- В процессе выполнения, когда пользовательское диалоговое окно отображается на экране. Для этого воспользуйтесь инструкциями VBA.

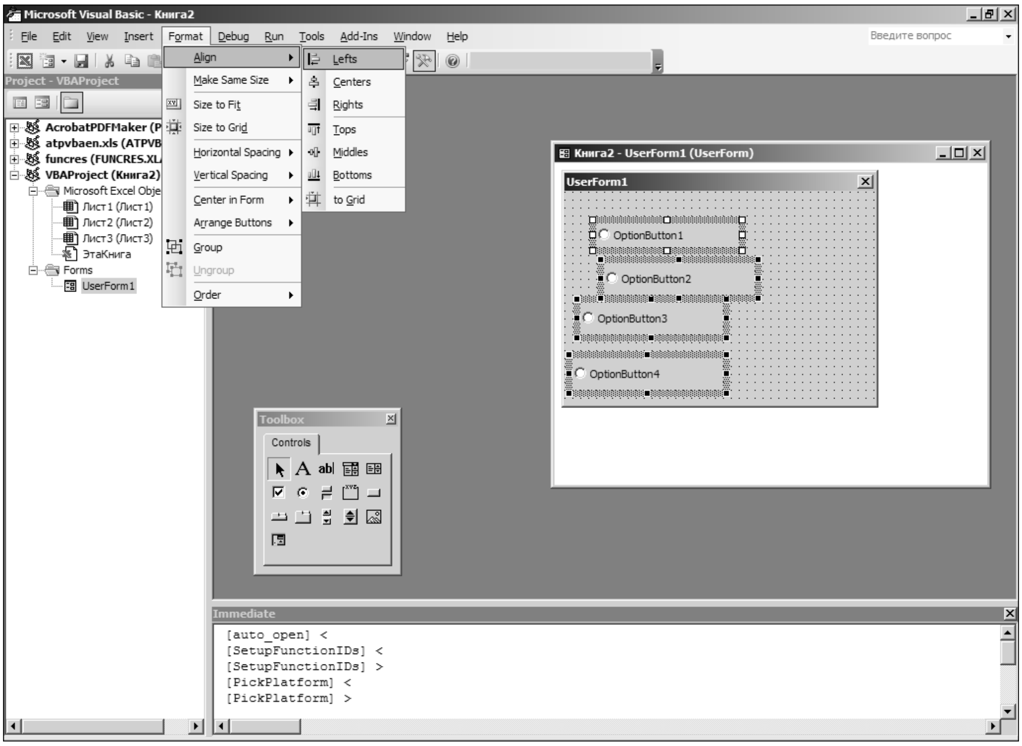


Рис. 13.3. Использование команды Format⇨Align *для изменения взаимного расположения элементов управления*

Использование окна Properties

В VBE окно Properties позволяет изменять свойства выделенного элемента управления (это может быть обычный элемент управления или сама форма `UserForm`). Кроме того, вы вправе выбрать элемент управления с помощью раскрывающегося списка в верхней части окна Properties (рис. 13.4).

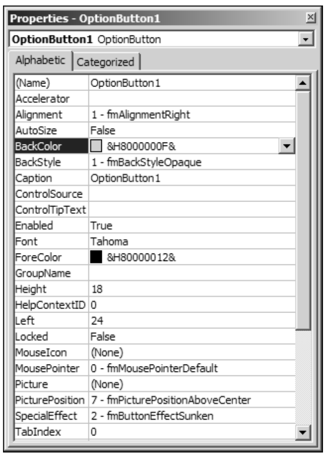


Рис. 13.4. Выбор элемента управления (`OptionButton`) из раскрывающегося списка в верхней части окна Properties

В окне Properties расположены две вкладки. Вкладка Alphabetic представляет свойства выделенного элемента управления в алфавитном порядке. Вкладка Categorized отображает свойства, сгруппированные в логические категории. Обе вкладки содержат одинаковые свойства, но в различном порядке.

Для того чтобы изменить свойство, необходимо щелкнуть на нем и ввести новое значение. Некоторые свойства могут принимать только ограниченный набор допустимых значений, выбираемых из соответствующего списка. При щелчке на таком свойстве в окне Properties будет отображена кнопка со стрелкой, указывающей вниз. Щелкните на этой кнопке, чтобы выбрать значение из предложенного списка. Например, свойство `TextAlign` может принимать одно из следующих значений: `1 - fmTextAlignLeft`, `2 - fmTextAlignCenter` и `3 - fmTextAlignRight`.

При выделении отдельных свойств (например, `Font` и `Picture`) рядом с ними отображается небольшая кнопка с троеточием. Щелчок на этой кнопке приводит к вызову диалогового окна настройки свойства.

Свойство `Picture` элемента управления `Image` ст*о*ит рассмотреть отдельно, поскольку для него необходимо указать графический файл. Еще один вариант — вставить изображение из буфера обмена. В последнем случае его сначала следует скопировать в буфер обмена, а затем выбрать свойство `Picture` элемента управления `Image` и нажать комбинацию клавиш <Ctrl+V> для вставки содержимого буфера обмена.

Если выделить два или больше элементов управления, то окно Properties будет отображать только те свойства, которые являются общими для выделенных объектов.

Объект `UserForm` характеризуется рядом свойств, значения которых можно изменять. Эти свойства используются в качестве значений, принятых по умолчанию для элементов управления, которые добавляются в пользовательское диалоговое окно. Например, если изменить свойство `Font` объекта `UserForm`, то все элементы управления, которые вставляются в пользовательское диалоговое окно, будут использовать указанный шрифт.

Общие свойства

Каждый элемент управления имеет как собственный набор уникальных свойств, так и ряд общих свойств, присущих другим элементам управления. Например, все элементы управления имеют свойство `Name` и свойства, определяющие его размер и расположение на форме (`Height`, `Width`, `Left` и `Right`).

Если вы собираетесь работать с элементом управления с помощью кода VBA, то ему стоит определить значащее имя. Например, первый элемент управления `OptionButton`, который добавлен в пользовательское диалоговое окно, по умолчанию получит имя `OptionButton1`. В коде ссылка на этот объект будет выглядеть следующим образом.

```
OptionButton1.Value = True
```

Но если элементу управления `OptionButton` присвоить описательное имя (например, `obLandscape`), то можно использовать такой оператор.

```
obLandscape.Value = True
```

Многие пользователи считают, что удобно использовать имена, которые указывают на тип объекта. В предыдущем примере был применен префикс `ob` для указания того, что объект представляет элемент управления `OptionButton`.

Можно изменять значения свойств нескольких элементов управления одновременно. Например, вы вправе создать на форме несколько элементов управления OptionButton и выровнять их все по левому краю. Для этого достаточно выделить все элементы управления OptionButton и изменить значение свойства Left в окне Properties. Все выделенные элементы управления примут новое значение свойства Left.

Получение дополнительной информации о свойствах

Диалоговое справочное руководство является лучшим способом получения информации о свойствах различных элементов управления. Щелкните на свойстве в окне Properties и нажмите клавишу <F1>. На рис. 13.5 показан пример справочных сведений, приведенных для выделенного свойства.

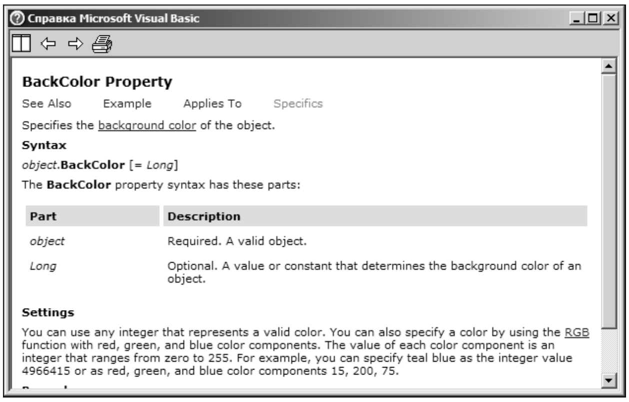


Рис. 13.5. Диалоговое справочное руководство предоставляет информацию о каждом свойстве элемента управления

Советы по использованию клавиатуры

Многие пользователи предпочитают перемещаться по диалоговым окнам с помощью клавиатуры. Комбинации клавиш <Tab> и <Shift+Tab> позволяют циклически переключаться между элементами управления. Чтобы удостовериться, что диалоговое окно корректно реагирует на команды с клавиатуры, обратите внимание на такие моменты: порядок просмотра элементов управления и комбинации клавиш.

Изменение порядка просмотра (активизации)

Порядок просмотра определяет последовательность, в которой активизируются элементы управления при нажатии пользователем комбинаций клавиш <Tab> и <Shift+Tab>. Кроме того, порядок активизации указывает, какой элемент управления по умолчанию выделяется на форме первым. Если пользователь вводит текст в элемент управления TextBox, то этот элемент управления считается активным. Если далее щелкнуть на элементе управления OptionButton, то именно он станет активным. Элемент управления, назначенный первым для просмотра, будет активным в момент открытия диалогового окна.

Для того чтобы указать порядок активизации, выберите команду View⇨Tab Order. Кроме того, можно щелкнуть правой кнопкой мыши на диалоговом окне и выбрать Tab Order из появившегося контекстного меню. В любом случае, Excel отобразит диалоговое окно Tab Order, которое показано на рис. 13.6. Диалоговое окно Tab Order содержит упорядоченный список всех элементов управления в последовательности, которая соответ-

ствует порядку активизации объектов в пользовательском диалоговом окне. Чтобы переместить элемент управления в списке, выберите его и щелкните на кнопке Move Up или Move Down. Можно одновременно перемещать более одного элемента управления (удерживайте при выделении элементов клавишу <Shift> или <Ctrl>).

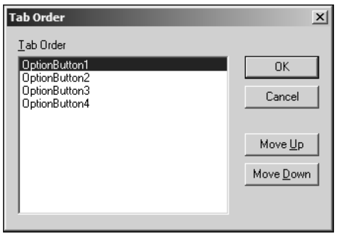


Рис. 13.6. Воспользуйтесь диалоговым окном Tab Order *для указания порядка просмотра (активизации) элементов управления*

С другой стороны, можно указать порядок активизации элемента управления с помощью окна Properties. Первый активизируемый элемент управления будет иметь свойство `TabIndex`, установленное в значение 0. Изменение значения свойства `TabIndex` текущего объекта приведет к изменению значений свойств `TabIndex` других элементов управления. Изменения вносятся автоматически. Вы можете удостовериться, что ни один элемент управления не имеет значение свойства `TabIndex` большее, чем количество элементов управления в диалоговом окне. Если необходимо удалить элемент управления из списка активизируемых объектов, то задайте его свойству `TabStop` значение `False`.

НА ЗАМЕТКУ

Некоторые элементы управления, такие как `Frame` и `MultiPage`, являются контейнерами для других элементов управления. Элементы управления в контейнере имеют собственный порядок активизации. Для того чтобы указать порядок активизации для элементов управления `OptionButton` внутри элемента управления `Frame`, выделите последний перед выбором команды View⇨Tab Order.

НАЗНАЧЕНИЕ КОМБИНАЦИЙ КЛАВИШ

Большинству элементов управления диалогового окна можно назначить комбинацию клавишу. Таким образом, пользователь получит доступ к элементу управления по нажатию <Alt> и указанной клавиши. Использование свойства `Accelerator` в окне Properties позволяет определить клавишу для активизации элемента управления.

СОВЕТ

Некоторые элементы управления не имеют свойства `Accelerator`, так как они не отображают значение свойства `Caption`. Но доступ посредством клавиатуры к таким элементам управления можно предоставить с помощью элемента управления `Label`. Назначьте клавишу элементу управления `Label`, после чего расположите его в последовательности активизации перед необходимым элементом управления `TextBox`.

Тестирование пользовательского диалогового окна

Обычно при разработке возникает необходимость в тестировании диалогового окна `UserForm`. Существует три способа, которые позволяют проверить диалоговое окно без вызова его из процедуры VBA.

- Выберите команду Run⇨Run Sub/UserForm.
- Нажмите клавишу <F5>.
- Щелкните на кнопке Run Sub/UserForm на панели инструментов Standard.

Эти методы позволяют запустить событие инициализации диалогового окна. Как только диалоговое окно будет отображено в тестовом режиме, можно начинать проверку порядка активизации объектов и поддержки комбинаций клавиш.

Отображение и закрытие пользовательского диалогового окна

В этом разделе предлагается обзор средств VBA, предназначенных для управления пользовательскими диалоговыми окнами.

Отображение пользовательского диалогового окна

Для того чтобы отобразить пользовательское диалоговое окно с помощью VBA, необходимо создать процедуру, которая вызывает метод `Show` объекта `UserForm`. Форму `UserForm` невозможно отобразить, не выполнив как минимум одну строку кода VBA. Если объект `UserForm` называется `UserForm1`, то следующая процедура отобразит это пользовательское диалоговое окно.

```
Sub ShowDialog()
    UserForm1.Show
End Sub
```

Данная процедура должна располагаться в стандартном модуле VBA, а не в модуле формы `UserForm`.

При отображении пользовательской формы она остается на экране до тех пор, пока ее не скроют. Обычно в пользовательскую форму добавляют элемент управления `CommandButton`, который запускает процедуру закрытия формы. Эта процедура либо выгружает пользовательскую форму с помощью метода `Unload`, либо скрывает пользовательскую форму с экрана с помощью метода `Hide` объекта `UserForm`. Детально с каждым из них вы познакомитесь в приведенных далее примерах.

В дополнение вы можете отобразить немодальную форму. В этом случае вы вправе продолжать работу в Excel, не скрывая саму форму. По умолчанию все пользовательские формы отображаются в модальном режиме (в нем нельзя выполнять редактирование данных Excel, не скрыв саму форму). Для отображения немодальной формы используется следующий синтаксис.

```
UserForm1.Show 0
```

ВНИМАНИЕ

В Excel версии до 2000 немодальные пользовательские формы не поддерживаются.

Если имя пользовательской формы сохраняется в строковой переменной, то с помощью метода `Add` вы можете добавить форму в коллекцию `UserForms`, а затем использовать метод `Show` этой коллекции. Ниже приведен пример назначения имени формы переменной `MyForm`, а также отображения ее на экране.

```
MyForm = "UserForm"
UserForms.Add(MyForm).Show
```

Последний способ применим в случаях использования в проекте нескольких форм и управления ими с помощью средств VBA.

Кроме того, VBA поддерживает оператор `Load`. Загрузка пользовательского диалогового окна приводит к сохранению объекта формы в памяти. Однако до тех пор, пока не будет выполнен метод `Show`, форма останется невидимой для остальной части программы. Для того чтобы загрузить диалоговое окно `UserForm1`, необходимо воспользоваться следующим оператором.

```
Load UserForm1
```

Если вы применяете сложное диалоговое окно, то вам может понадобиться предварительно загрузить его в память, чтобы в случае необходимости быстро отобразить методом `Show`. Как правило, использование метода `Load` не имеет смысла.

Закрытие пользовательского диалогового окна

Для того чтобы закрыть форму `UserForm1`, воспользуйтесь командой `Unload`.

```
Unload UserForm1
```

Также можно применить следующий оператор.

```
Unload Me
```

В этом случае ключевое слово `Me` применяется для идентификации пользовательской формы.

Обычно в коде VBA команда `Unload` выполняется только после того, как форма `UserForm` выполнит все свои функции. Например, форма `UserForm` может содержать элемент управления `CommandButton`, который используется в качестве кнопки OK. Щелчок на этой кнопке приводит к выполнению заранее определенного макроса. Один из операторов макроса заключается в выгрузке формы `UserForm` из памяти. В результате пользовательское диалоговое окно будет отображаться на экране до тех пор, пока макрос, содержащий оператор `Unload`, не завершит свою работу.

Когда форма `UserForm` выгружается из памяти, элементы управления, содержавшиеся на ней, возвращаются в первоначальное состояние. Другими словами, в коде нельзя обращаться к значениям, указываемым пользователем, после того как форма выгружена из памяти. Если значения, введенные пользователем, будут применяться позже (после выгрузки диалогового окна `UserForm`), то необходимо сохранить их в переменной с областью действия `Public`, которая определена в стандартном модуле VBA. Кроме того, значение всегда можно сохранить в ячейке листа.

Диалоговое окно автоматически выгружается из памяти, когда пользователь щелкает на кнопке Отмена (Кнопка × в строке заголовке окна). Это действие приводит к возникновению события `QueryClose` объекта `UserForm`, после которого генерируется событие `Terminate` объекта `UserForm`.

Объект `UserForm` может использовать метод `Hide`. При его вызове диалоговое окно исчезает, но остается в памяти, поэтому в коде можно получить доступ к различным свойствам элементов управления. Ниже приведен пример оператора, который скрывает диалоговое окно.

```
UserForm1.Hide
```

Также можно воспользоваться следующим оператором.

```
Me.Hide
```

Если по какой-либо причине необходимо, чтобы пользовательское диалоговое окно было немедленно скрыто в процессе выполнения макроса, воспользуйтесь методом `Hide` в самом начале процедуры, а затем укажите команду `DoEvents`. Например, в следующей процедуре форма `UserForm` немедленно исчезнет после того, как пользователь щелкнет на кнопке `CommandButton1`. Последний оператор процедуры выгружает пользовательское диалоговое окно из памяти.

```
Private Sub CommandButton1_Click()
    Me.Hide
    DoEvents
    For r = 1 To 10000
```

```
        Cells(r, 1) = r
    Next r
    Unload Me
End Sub
```

В главе 15 показано, как отображать индикатор прогресса выполнения макроса, основу которого составляет условие: пользовательское окно отображается на экране до тех пор, пока выполняется макрос.

О процедурах обработки событий

Как только диалоговое окно появляется на экране, пользователь начинает с ним взаимодействовать — выбирать опции в элементе управления `ListBox`, щелкать на кнопках `CommandButton` и т.д. Используя официальную терминологию, можно сказать, что пользователь генерирует события. Например, щелчок на элементе управления `CommandButton` приводит к возникновению события `Click` объекта `CommandButton`. Вам необходимо создать процедуры, которые будут выполняться при возникновении соответствующих событий. Первые называются *обработчиками событий*.

Процедуры обработки событий вводятся в модуле кода объекта `UserForm`. Наряду с этим процедура обработки события может вызывать другие процедуры, которые находятся в стандартном модуле VBA.

В коде VBA можно изменять свойства элементов управления, пока пользовательское диалоговое окно отображается на экране (т.е. на этапе выполнения). Например, вы можете назначить элементу управления `ListBox` процедуру, которая изменяет текст элемента управления `Label` при выборе элемента списка. Подобное поведение элементов управления подробно рассмотрено далее в этой главе.

Пример создания пользовательского диалогового окна

Если раньше вы никогда не создавали пользовательские диалоговые окна, то обратите внимание на пример, приведенный в этой главе. В нем вы найдете пошаговые инструкции по созданию простого диалогового окна и разработке процедуры VBA для поддержки этого диалогового окна.

В приведенном примере представлено диалоговое окно, предназначенное для получения следующей информации: имени и пола пользователя. В диалоговом окне вы найдете элемент управления `TextBox`, используемый для введения имени, и три элемента управления `OptionButton` для указания пола (мужчина, женщина и другое). Информация, полученная в диалоговом окне, заносится в пустую строку рабочего листа.

Создание пользовательского диалогового окна

На рис. 13.7 отображено завершенное диалоговое окно приведенного примера. Чтобы получить наилучший результат, создайте новую рабочую книгу с одним рабочим листом. После этого следуйте представленным далее инструкциям.

1. Нажмите комбинацию клавиш <Alt+F11> для запуска VBE.
2. В окне Project укажите проект рабочей книги, а затем выберите Insert⇨UserForm, чтобы добавить пустое диалоговое окно.

3. Свойство `Caption` объекта `UserForm` будет иметь значение по умолчанию — `UserForm1`. Воспользуйтесь окном Properties, чтобы изменить значение свойства `Caption` на `Введите свое имя и пол` (если окно Properties не отображается на экране, нажмите клавишу <F4>).

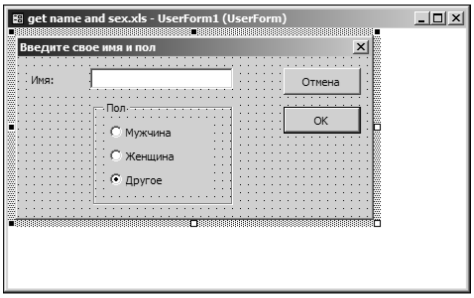


Рис. 13.7. Это диалоговое окно запрашивает у пользователя его имя и пол

4. Добавьте на форму элемент управления `Label` и измените его свойства, как показано ниже.

Свойство	Значение
`Accelerator`	N
`Caption`	Имя:
`TabIndex`	0

5. Добавьте элемент управления `TextBox` и измените его свойства следующим образом.

Свойство	Значение
`Name`	TextName
`TabIndex`	1

6. Добавьте элемент управления `Frame` и измените его свойства.

Свойство	Значение
`Caption`	Пол
`TabIndex`	2

7. Добавьте элемент управления `OptionButton` внутри элемента управления `Frame` и измените его свойства, как показано ниже.

Свойство	Значение
`Accelerator`	M
`Caption`	Мужчина
`Name`	OptionMale
`TabIndex`	0

8. Добавьте в элемент управления Frame еще один элемент управления OptionButton и измените его свойства так, как представлено далее.

Свойство	Значение
Accelerator	F
Caption	Женщина
Name	OptionFemale
TabIndex	1

9. Добавьте в элемент управления Frame еще один элемент управления OptionButton и измените его свойства следующим образом.

Свойство	Значение
Accelerator	U
Caption	Другое
Name	OptionUnknown
TabIndex	2
Value	True

10. Добавьте элемент управления CommandButton за пределами элемента управления Frame и измените его свойства.

Свойство	Значение
Caption	OK
Default	True
Name	OKButton
TabIndex	3

11. Добавьте элемент управления CommandButton за пределами элемента управления Frame и измените его свойства, как показано ниже.

Свойство	Значение
Caption	Отмена
Default	True
Name	CancelButton
TabIndex	4

СОВЕТ

При создании нескольких подобных элементов управления может оказаться, что быстрее копировать существующий элемент управления, чем добавлять новый. Для того чтобы скопировать элемент управления, удерживайте клавишу <Ctrl> при перетаскивании элемента управления, что приведет к созданию копии. После этого вам останется внести изменения в свойства созданной копии объекта.

Создание кода для отображения диалогового окна

После создания элементов управления на лист необходимо добавить объект `CommandButton`. Эта кнопка будет запускать процедуру, которая предназначена для отображения формы `UserForm`. Для этого выполните следующие действия.

1. Перейдите в окно Excel (воспользуйтесь комбинацией клавиш <Alt+F11>).
2. Щелкните правой кнопкой мыши на любой панели инструментов и выберите Элементы управления из появившегося контекстного меню. Excel отобразит соответствующую панель инструментов на экране. Данная панель подобна панели Toolbox в VBE.
3. Воспользуйтесь панелью инструментов Элементы управления, чтобы добавить на лист объект Кнопка. Щелкните на значке Кнопка, после этого перетащите его на лист для создания кнопки.
4. Измените при необходимости подпись объекта Кнопка. Для этого щелкните правой кнопкой мыши на объекте Кнопка и выберите Объект CommandButton⇨Edit из появившегося контекстного меню. После этого отредактируйте текст, который отображается на кнопке.
5. Дважды щелкните на элементе управления Кнопка.
6. Это приведет к активизации VBE. В нем отображается модуль кода для листа с открытой пустой процедурой обработки событий объекта `CommandButton` (Кнопка), который расположен на рабочем листе.
7. Введите в процедуру `CommandButton1_Click` единственный оператор (рис. 13.8). Эта короткая процедура вызывает метод `Show` объекта `UserForm1` для отображения на экране пользовательского диалогового окна `UserForm`.

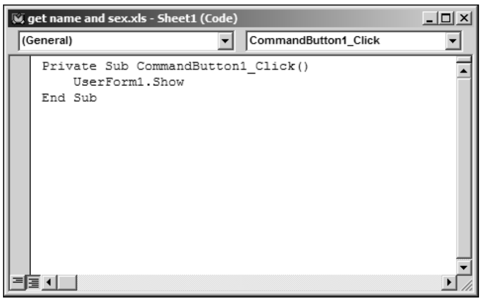


Рис. 13.8. Процедура `CommandButton1_Click` выполняется после щелчка на кнопке, расположенной на рабочем листе

Проверка

Следующим этапом является проверка процедуры, отображающей диалоговое окно.

НА ЗАМЕТКУ

После щелчка на кнопке на рабочем листе ничего не произойдет. Вернее, кнопка будет выделена, но никакие действия она не инициирует. Это связано с тем, что Excel все еще находится в режиме конструктора, в который она автоматически переходит каждый раз, когда с помощью панели инструментов Элементы управления на лист добавляется новый элемент управления. Для того чтобы покинуть режим конструктора, щелкните на кнопке Выход из режима конструктора на панели инструментов Элементы управления.

Как только Excel выйдет из режима конструктора, щелчок на кнопке приведет к отображению пользовательского диалогового окна (рис. 13.9).

Когда диалоговое окно будет отображено, введите произвольный текст в текстовом поле и щелкните на кнопке OK. В результате — ничего не происходит: что совершенно справедливо, так как для объекта `UserForm` не создано ни одной процедуры обработки событий.



Щелчок на кнопке × в строке заголовка диалогового окна позволит закрыть его.

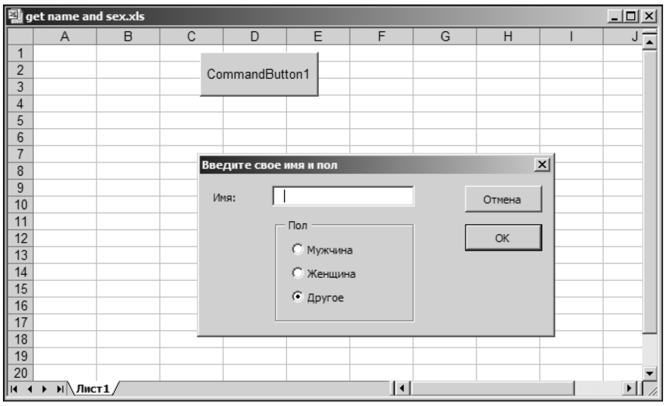


Рис. 13.9. Событие `Click` объекта `CommandButton` приводит к отображению пользовательского диалогового окна

Добавление процедур обработки событий

В этом разделе рассматривается задача создания процедур, которые обрабатывают события, возникающие после открытия пользовательского диалогового окна. Выполните следующие действия.

1. Нажмите комбинацию клавиш <Alt+F11>. Это приведет к активизации VBE.
2. Удостоверьтесь, что пользовательское диалоговое окно отображено на экране и, если это так, дважды щелкните на кнопке Отмена. Таким образом, будет активизировано окно кода для объекта кнопки формы `UserForm`, также будет добавлена пустая процедура — `CancelButton_Click`. Обратите внимание, что название процедуры состоит из имени объекта, символа подчеркивания и названия события, которое обрабатывает процедура.
3. Модифицируйте процедуру, как показано ниже (это обработчик события `Click` объекта `CancelButton`).

```
Private Sub CancelButton_Click()
    Unload UserForm1
End Sub
```

4. Данная процедура выполняется после щелчка пользователем на кнопке Отмена. Она вызывает выгрузку формы `UserForm` из памяти.
5. Нажмите комбинацию клавиш <Shift+F7>, чтобы повторно отобразить объект `UserForm1` (или щелкните на значке View Object в верхней части окна Project Explorer).

6. Дважды щелкните на кнопке OK и введите следующую процедуру (это обработчик события Click объекта OKButton).

```
Private Sub OKButton_Click()
'   Активизация листа
    Sheets("Лист1").Activate

'   Определение следующей пустой строки
    NextRow = _
      Application.WorksheetFunction.CountA(Range("A:A")) + 1
'   Передача имени
    Cells(NextRow, 1) = TextName.Text

'   Передача пола
    If OptionMale Then Cells(NextRow, 2) = "Мужчина"
    If OptionFemale Then Cells(NextRow, 2) = "Женщина"
    If OptionUnknown Then Cells(NextRow, 2) = "Другое"

'   Очистка элементов управления для следующих записей
    TextName.Text = ""
    OptionUnknown = True
    TextName.SetFocus
End Sub
```

7. Перейдите в окно Excel и щелкните на кнопке еще раз, чтобы отобразить пользовательское диалоговое окно. Запустите процедуру повторно.
8. Элементы управления диалогового окна должны функционировать правильно. На рис. 13.10 показан результат правильного поведения диалогового окна.

Процедура OKButton_Click работает следующим образом: сначала она проверяет, активен ли лист Лист1. После этого запускается функция Excel СЧЕТ (COUNT) для определения следующей пустой ячейки в столбце А. Затем текст из текстового поля TextBox передается в определенную ячейку столбца A. С помощью операторов If определяется выделенный элемент управления OptionButton, что обеспечивает запись соответствующего текста в столбец B (Мужчина, Женщина, Другое). Наконец, диалоговое окно перезапускается (чтобы обеспечить возможность введения следующей записи). Заметим, что щелчок на кнопке OK не приведет к закрытию диалогового окна. Для завершения ввода данных (и выгрузки пользовательского диалогового окна) щелкните на кнопке Отмена.

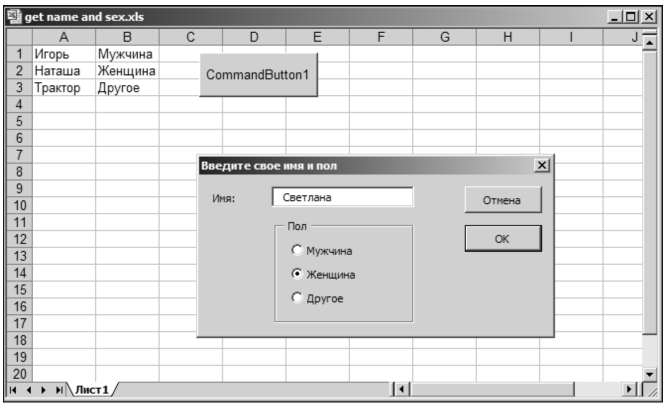


Рис. 13.10. Вызов пользовательского диалогового окна

Проверка правильности введенных данных

Приведенному в этом разделе примеру следует уделить особое внимание. Возможно, вы заметили, что не устранена небольшая проблема — отсутствует проверка введенных в текстовом поле данных (вы не знаете, ввел ли пользователь свое имя). Следующий код добавлен в процедуру `OKButton_Click` перед оператором вставки текста на рабочий лист. Он проверяет, ввел ли пользователь свое имя (на самом деле проверяется наличие любого текста) в поле `TextBox`. Если текстовое поле `TextBox` осталось пустым, то выводится соответствующее сообщение, и текстовое поле снова становится активным. Таким образом, пользователь сможет приступить к введению своего имени. Оператор `Exit Sub` завершает выполнение процедуры без выполнения дополнительных действий.

```
'   Проверка введения имени
    If TextName.Text = "" Then
        MsgBox "Введите имя"
        TextName.SetFocus
        Exit Sub
    End If
```

Заработало!

После внесения соответствующих исправлений диалоговое окно будет работать безупречно (не забудьте проверить работоспособность комбинаций клавиш). В реальной жизни вам может потребоваться собрать много дополнительной информации, а не только сведений об имени и поле пользователя. Но в любом случае вы должны применять изложенные выше принципы. Вам останется добавить в диалоговое окно большее количество элементов управления.

События объекта UserForm

Каждый элемент управления в форме `UserForm` (а также сам объект `UserForm`) разрабатываются, чтобы реагировать на определенные события. Эти события возникают в результате действий пользователя или генерируются программой Excel. Например, щелчок на кнопке `CommandButton` приводит к возникновению события `Click` объекта `CommandButton`. Можно создать код, который будет выполняться при возникновении определенного события.

Некоторые действия приводят к возникновению одновременно нескольких событий. Например, щелчок на кнопке со стрелкой “вверх” в элементе управления `SpinButton` приведет к возникновению события `SpinUp` и события `Change`. После того, как пользовательское диалоговое окно будет отображено с помощью метода `Show`, Excel сгенерирует события `Initialize` и `Activate` объекта `UserForm`.

Кроме того, Excel поддерживает события объектов `Sheet` (`Лист`), `Chart` (`Диаграмма`) и `ThisWorkbook` (`ЭтаКнига`). Эти типы событий рассматриваются в главе 18.

Получение дополнительной информации о событиях

Для того чтобы получить информацию о событиях, которые генерируются конкретным элементом управления, выполните следующие действия.

1. Добавьте элемент управления в пользовательское диалоговое окно.

2. Дважды щелкните на элементе управления, чтобы открыть модуль кода для объекта `UserForm`. VBE вставит пустую процедуру обработки события, принятого по умолчанию.
3. Щелкните на раскрывающемся списке в правом верхнем углу окна модуля и просмотрите полный список событий, которые поддерживаются текущим элементом управления (рис. 13.11).

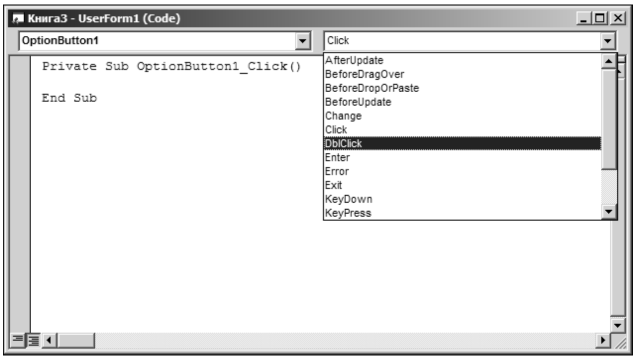


Рис. 13.11. Список событий для элемента управления `Option Button`

4. Выберите событие из списка, и VBE создаст пустой обработчик события.



Для того чтобы получить информацию о событии, обратитесь к диалоговой справочной системе. В справочной системе находится описание событий, которые поддерживаются каждым элементом управления.



Имя процедуры обработки событий содержит полное имя того объекта, который генерирует событие. Таким образом, если изменить имя элемента управления, то необходимо внести соответствующие изменения и в имя процедуры обработки события. Изменение имени процедуры автоматически не выполняется! Для того чтобы облегчить собственную жизнь, присвойте описательные имена элементам управления до того, как приступите к созданию процедуры обработки соответствующих событий.

События объекта UserForm

Несколько событий непосредственно связано с отображением и выгрузкой объекта `UserForm`.

- `Initialize` — происходит перед загрузкой и отображением формы `UserForm`. Не происходит, если объект `UserForm` до этого был скрыт.
- `Activate` — происходит в момент отображения объекта `UserForm`.
- `Deactivate` — происходит в момент деактивизации объекта `UserForm`. Не происходит при скрытии формы `UserForm`.
- `QueryClose` — происходит перед выгрузкой объекта `UserForm`.
- `Terminate` — происходит после выгрузки объекта `UserForm`.

Довольно важно правильно выбрать подходящее событие для процедуры обработки события и проанализировать порядок генерирования событий. Использование метода `Show` приводит к возникновению событий `Initialize` и `Activate` (в указанном порядке). Результатом выполнения метода `Load` является только генерирование события `Initialize`. Применяя метод `Unload`, вы вызываете появление событий `QueryClose` и `Terminate` (в указанном порядке). Метод `Hide` не генерирует ни одно из перечисленных событий.

На прилагаемом к книге компакт-диске находится рабочая книга, которая управляет описанными событиями и отображает в момент возникновения события специальное сообщение. Если изучение событий объекта `UserForm` у вас связано с большими трудностями, то, проанализировав код этого примера, вы сможете ответить на многие вопросы.

События элемента управления SpinButton

Для того чтобы разобраться с концепцией событий, в этом разделе мы подробно рассмотрим события, связанные с элементом управления `SpinButton`.

На прилагаемом к книге компакт-диске содержится рабочая книга, которая демонстрирует применение событий, генерируемых объектами `SpinButton` и `UserForm` (первый содержится во втором). Рабочая книга включает несколько процедур обработки событий — по одной на каждое событие элемента управления `SpinButton` и объекта `UserForm`. Каждая из этих процедур создает окно сообщения, которое указывает на возникновение соответствующего события.

В табл. 13.1 перечислены все события, связанные с элементом управления `SpinButton`.

Таблица 13.1. События объекта SpinButton

Событие	Описание
`AfterUpdate`	Происходит после того, как элемент управления изменяется с помощью пользовательского интерфейса
`BeforeDragOver`	Происходит в процессе операции перетаскивания объекта
`BeforeDrop-OrPaste`	Происходит перед тем, как пользователь отпустит перетаскиваемый объект или скопирует его из буфера обмена
`BeforeUpdate`	Происходит перед изменением элемента управления
`Change`	Происходит в момент изменения значения свойства `Value`
`Enter`	Происходит перед тем, как элемент управления `SpinButton` будет активизирован после другого элемента управления этой же формы `UserForm`
`Error`	Происходит в момент обнаружения элементом управления ошибки; при этом элемент управления не сможет передать информацию об ошибке в вызывающую программу
`Exit`	Происходит непосредственно перед тем, как элемент управления деактивизируется, активным становится другой элемент управления текущей формы
`KeyDown`	Происходит, когда пользователь нажимает клавишу при активном объекте
`KeyPress`	Происходит, когда пользователь нажимает клавишу по вводу символа
`KeyUp`	Происходит, когда пользователь отпускает клавишу и объект активный
`SpinDown`	Происходит, когда пользователь щелкает на нижней (или левой) кнопке элемента управления `SpinButton`
`SpinUp`	Происходит, когда пользователь щелкает на верхней (или правой) кнопке элемента управления `SpinButton`

Пользователь может управлять объектом SpinButton с помощью мыши или (если элемент управления активен) клавиш управления курсором.

События мыши

Когда пользователь щелкает на верхней кнопке элемента управления SpinButton, происходят следующие события.

1. Enter (генерируется только в том случае, если элемент управления неактивен).
2. Change.
3. SpinUp.

События клавиатуры

Пользователь может нажать клавишу <Tab> для того, чтобы сделать активным элемент управления SpinButton. Только после этого можно использовать клавиши управления курсором для изменения значения элемента управления. Если все именно так и происходит, то события генерируются в следующем порядке.

1. Enter.
2. KeyDown.
3. Change.
4. SpinUp.

События, генерируемые кодом?

Элемент управления SpinButton может изменяться в коде VBA, что также провоцирует возникновение соответствующих событий. Например, представленный далее оператор устанавливает свойства Value элемента управления SpinButton в значение 0, а это приводит к возникновению события Change. Такой результат достигается только в том случае, если исходное свойство Value не равно нулю.

```
SpinButton1.Value = 0
```

Вы вправе предположить, что выполнить отмену генерирования событий можно, установив свойство EnableEvents объекта Application в значение False. К сожалению, это свойство поддерживается только объектами, которые являются “истинными” в Excel: Workbook, Worksheet и Chart.

Совместное использование элементов управления SpinButton и TextBox

Элемент управления SpinButton имеет свойство Value, но не имеет возможности отображать значение этого свойства. В большинстве случаев необходимо, чтобы пользователь мог изменить значение элемента управления SpinButton непосредственно, а не многократно щелкая на элементе управления.

Эффективным решением может стать объединение элемента управления SpinButton с элементом управления TextBox, что позволяет пользователю вводить значение элемента управления SpinButton непосредственно, используя для этого поле элемента управления TextBox. Кроме того, щелчок на элементе управления SpinButton позволит изменить значение, отображаемое в элементе управления TextBox.

На рис. 13.12 приведен простой пример. Свойство Min элемента управления SpinButton имеет значение 1, а свойство Max — 100. Таким образом, щелчок на одной из стрелок элемента управления SpinButton приведет к изменению значения в пределах от 1 до 100.

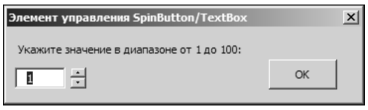


Рис. 13.12. Элемент управления `SpinButton`, совмещенный с элементом управления `TextBox`

Эта рабочая книга находится на прилагаемом к книге компакт-диске.

Код, реализующий “связывание” элементов управления SpinButton и TextBox, достаточно простой. В целом все сводится к созданию процедур обработки событий, которые будут синхронизировать свойство Value элемента управления SpinButton и свойство Text элемента управления TextBox.

Представленная далее процедура выполняется каждый раз при возникновении события Change элемента управления SpinButton. Таким образом, процедура выполняется тогда, когда пользователь щелкает на элементе управления SpinButton или изменяет его значение, нажав одну из клавиш управления курсором.

```
Private Sub SpinButton1_Change()
    TextBox1.Text = SpinButton1.Value
End Sub
```

Процедура просто приравнивает значение свойства Value элемента управления SpinButton к свойству Text элемента управления TextBox. В данном случае элементы управления имеют имена, заданные по умолчанию (SpinButton1 и TextBox1). Если пользователь введет значение непосредственно в элемент управления TextBox, то будет сгенерировано событие Change, после чего должен выполняться следующий код.

```
Private Sub TextBox1_Change()
    NewVal = Val(TextBox1.Text)
    If NewVal >= SpinButton1.Min And _
        NewVal <= SpinButton1.Max Then _
        SpinButton1.Value = NewVal
End Sub
```

Эта процедура начинается с вызова функции VBA Val, которая преобразует текст элемента управления TextBox в числовое значение (если элемент управления TextBox содержит строку, то функция Val возвращает значение 0). Следующий оператор определяет, попадает ли значение в указанный диапазон допустимых значений. Если это так, то свойство Value элемента управления SpinButton устанавливается в значение, которое введено в поле элемента управления TextBox.

Пример организован таким образом, что щелчок на кнопке OK (которая называется OKButton) передает значение элемента управления SpinButton в активную ячейку. Процедура обработки события Click элемента управления CommandButton выглядит следующим образом.

```
Private Sub OKButton_Click()
'   Введение значения в активную ячейку
    If CStr(SpinButton1.Value) = TextBox1.Text Then
        ActiveCell = SpinButton1.Value
        Unload Me
    Else
        MsgBox "Неправильное значение.", vbCritical
        TextBox1.SetFocus
        TextBox1.SelStart = 0
        TextBox1.SelLength = Len(TextBox1.Text)
    End If
End Sub
```

Данная процедура проводит последнюю проверку: анализируется текст, введенный в поле элемента управления TextBox, и значения элемента управления SpinButton. Такая процедура обрабатывает ситуации неверного ввода данных. Например, если пользователь введет в поле элемента управления TextBox текст 3r, то значение элемента управления SpinButton не изменится, а результат, который помещается в активную ячейку, будет отличным от ожидаемого. Обратите внимание, что значение свойства Value элемента управления SpinButton преобразуется в строку с помощью функции CStr. Это позволяет предотвратить ошибку, которая возникает, когда числовое значение сравнивается с текстовым. Если значение элемента управления SpinButton не соответствует содержимому элемента управления TextBox, то на экране отображается специальное сообщение. Причем объект TextBox активен, а его содержимое — выделено (с помощью свойств SelStart и SelLength). Таким образом, пользователю проще исправить неправильные значения.

О свойстве Tag

Каждый объект UserForm и каждый элемент управления имеют свойство Tag. Это свойство не представляет конечные данные и по умолчанию не имеет значения. Свойство Tag можно использовать для хранения информации, которая будет применена в программе.

Например, можно создать набор объектов TextBox в пользовательском диалоговом окне. От пользователя требуется ввести текст только в некоторые из них. В отдельные поля вводить текст не обязательно. Можно использовать свойство Tag для идентификации полей, которые необходимо заполнять. В этом случае значение свойства Tag — это строка, например, Required. Поэтому при написании кода обработки введенных пользователем данных можно ссылаться на свойство Tag.

Пример, приведенный ниже, представляет функцию, которая проверяет все элементы управления TextBox объекта UserForm1 и возвращает количество пустых текстовых полей, “требующих” введения информации.

```
Function EmptyCount()
  Dim ctl As Control
  EmptyCount= 0
  For Each ctl In UserForm1.Controls
    If TypeName(ctl) = "TextBox" Then
      If ctl.Tag = "Required" Then
        If ctl.Text = "" Then
           EmptyCount = EmptyCount + 1
        End If
      End If
   End If
 Next ctl
End Function
```

При работе с пользовательскими диалоговыми окнами можно придумать и другое достойное применение для свойства Tag.

Ссылка на элементы управления пользовательского диалогового окна

При работе с элементами управления, находящимися в форме `UserForm`, код VBA обычно содержится в модуле кода объекта `UserForm`. Кроме того, на элементы управления диалогового окна можно ссылаться из модуля кода VBA общего назначения. Для выполнения этой задачи необходимо задать правильную ссылку на элемент управления, указав имя объекта `UserForm`. В качестве примера рассмотрим процедуру, которая введена в модуле кода VBA. Эта процедура отображает пользовательское диалоговое окно, которое называется `UserForm1`.

```
Sub GetData()
    UserForm1.Show
End Sub
```

Предполагается, что диалоговое окно `UserForm1` содержит текстовое поле (`TextBox1`). Вам же необходимо указать значение текстового поля по умолчанию. Процедуру можно изменить следующим образом.

```
Sub GetData()
    UserForm1.TextBox1.Value = "Джон"
    UserForm1.Show
End Sub
```

Еще одним способом установки значения по умолчанию является использование события `Initialize` объекта `UserForm`. Можно написать код процедуры `UserForm_Initialize`, который будет располагаться в модуле кода диалогового окна.

```
Private Sub UserForm_Initialize()
    TextBox1.Value = "Джон"
End Sub
```

Обратите внимание, что при обращении к элементу управления из модуля кода диалогового окна не обязательно вводить в ссылку имя объекта `UserForm`. Подобное определение ссылок на элементы управления имеет свое преимущество: всегда можно воспользоваться средством Auto List Member, которое позволяет выбирать имена элементов управления из раскрывающегося списка. Вместо того чтобы использовать фактическое имя объекта `UserForm`, предпочтительнее применить имя `Me`. В противном случае, если имя объекта `UserForm` изменится, то вам не потребуется изменять все ссылки (с его участием) в коде.

Использование коллекций элементов управления

Элементы управления пользовательских диалоговых окон составляют отдельную коллекцию. Например, следующий оператор отображает количество элементов управления в диалоговом окне `UserForm1`.

```
MsgBox UserForm1.Controls.Count
```

Коллекции не представлены для каждого типа элементов управления. Например, не существует коллекции элементов управления `CommandButton`. Но тип элемента управления можно определить с помощью функции `TypeName`. Следующая процедура использует структуру `For Each` для циклического просмотра элементов коллекции `Controls`. В результате отображается количество элементов управления `CommandButton`, которые входят в коллекцию элементов управления объекта `UserForm1`.

```
Sub CountButtons()
    Dim cbCount As Integer
    Dim ctl as Control
    cbCount = 0
    For Each ctl In UserForm1.Controls
        If TypeName(ctl) = "CommandButton" Then _
          cbCount = cbCount + 1
    Next ctl
    MsgBox cbCount
End Sub
```

Настройка панели инструментов Toolbox

Если объект UserForm активен в редакторе VBE, панель Toolbox отображает элементы управления, которые можно добавить в пользовательское диалоговое окно. В этом разделе рассматриваются способы настройки окна Toolbox.

Изменение значков или текста подсказок

Если для определенного инструмента предпочтительнее использовать другой значок или текст подсказки, щелкните правой кнопкой мыши на соответствующем инструменте и выберите Customize Control из появившегося контекстного меню. На экране будет отображено диалоговое окно, которое позволяет изменить экранную подсказку или значок и загрузить новую пиктограмму из файла.

Добавление новых страниц

Окно Toolbox изначально содержит одну вкладку. Щелкните на ней правой кнопкой мыши и выберите New Page, чтобы добавить новую вкладку в окно Toolbox. Кроме того, можно изменить текст, который отображается на вкладке. Для этого выберите опцию Rename из контекстного меню.

Настройка или комбинирование элементов управления

Рекомендуем предварительно настроить элементы управления и сохранить их для дальнейшего использования. Можно, например, создать элемент управления CommandButton, который настроен на выполнение роли кнопки OK. Допускается изменять следующие параметры кнопки: Width (Ширина), Height (Высота), Caption (Подпись), Default (По умолчанию) и Name (Имя). После этого перетащите модифицированный элемент управления CommandButton на панель инструментов Toolbox. Это приведет с созданию нового элемента управления. Щелкните на элементе управления правой кнопкой мыши, чтобы переименовать его или изменить значок.

Также можно создать новый раздел окна Toolbox, который будет содержать несколько элементов управления. Например, вы вправе создать два элемента управления CommandButton, которые будут представлять кнопки OK и Отмена. Настройте их так, как это необходимо. Затем выберите обе кнопки и переместите их в окно панели инструментов Toolbox. Впоследствии можно использовать новый элемент управления окна Toolbox для быстрого создания необходимых кнопок.

Этот метод также применим к элементами управления, которые используются в качестве контейнера. Например, создайте элемент управления Frame и добавьте в него четыре модифицированных элемента управления OptionButton (соответствую-

щим образом расположив их на форме). После этого перетащите элемент управления `Frame` на панель инструментов Toolbox, чтобы создать модифицированный элемент управления `Frame`.

Порой требуется разместить модифицированные элементы управления на отдельной вкладке окна Toolbox. Таким образом, появляется возможность экспортировать вкладку окна Toolbox для совместного использования элементов управления другими пользователями Excel. Для того чтобы экспортировать вкладку окна Toolbox, щелкните на ней правой кнопкой мыши и выберите Export Page.

На прилагаемом к книге компакт-диске содержится файл `.PAG` с некоторыми модифицированными элементами управления. Можно импортировать этот файл в качестве новой вкладки окна Toolbox. Щелкните правой кнопкой мыши на вкладке и выберите команду Import Page. После этого укажите расположение файла `.PAG`. В результате окно Toolbox будет напоминать показанное на рис. 13.13.



Рис. 13.13. Окно Toolbox, содержащее новую вкладку с элементами управления

Добавление элементов управления ActiveX

Пользовательское диалоговое окно содержит также другие элементы управления ActiveX, разработанные компанией Microsoft и независимыми производителями. Для того чтобы добавить дополнительные элементы управления ActiveX на панель инструментов, щелкните правой кнопкой мыши в окне Toolbox и выберите Additional Controls. В результате будет отображено диалоговое окно, показанное на рис. 13.14.

Диалоговое окно Additional Controls содержит все элементы управления ActiveX, установленные в системе. Выберите элементы управления, которые необходимо вставить на панель инструментов. После этого щелкните на кнопке OK для добавления значков каждого из выбранных элементов управления.

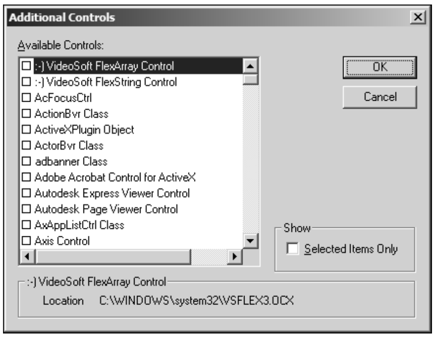


Рис. 13.14. Диалоговое окно Additional Controls позволяет добавлять элементы управления ActiveX

Не все элементы управления ActiveX, которые установлены в системе, поддерживаются пользовательскими диалоговыми окнами Excel. Скажем более — не поддерживается бо́льшая их часть. Кроме того, некоторые элементы управления требуют наличия лицензии на использование в собственных приложениях. Если пользователи приложения не имеют лицензии на использование определенного элемента управления, то на экране появится сообщение об ошибке.

Эмулирование диалоговых окон Excel

Внешний вид и поведение диалоговых окон Windows изменяется от программы к программе. При разработке приложений для Excel рекомендуется следовать стилю диалоговых окон Excel всегда, когда это возможно.

Наилучшим методом изучения способов создания эффективных диалоговых окон является повторное создание одного из стандартных диалоговых окон Excel. Например, удостоверьтесь, что вы правильно определили комбинации клавиш и активизировали элементы управления. Для того чтобы создать копию одного диалогового окна Excel, необходимо протестировать его в различных условиях. Один только анализ диалоговых окон Excel поможет улучшить познания в вопросах структуры окон и методов создания элементов управления.

Со временем вы убедитесь, что невозможно повторить отдельные диалоговые окна Excel, даже с помощью VBA. Например, при использовании кнопки Найти все в диалоговом окне Найти и заменить в Excel диалоговое окно изменяет свой размер. Пользовательское же диалоговое окно может иметь только постоянный размер.

Создание шаблонов диалоговых окон

Зачастую случается так, что при создании пользовательского диалогового окна каждый раз на форму добавляются одни и те же элементы управления. Например, все пользовательские диалоговые окна имеют два элемента управления `CommandButton`, используемых в качестве кнопок OK и Отмена. В предыдущем разделе рассматривались методы комбинирования элементов управления с целью получения одного элемента управления, обладающего функциями двух. Еще одной программной уловкой может служить шаблон диалогового окна, который при необходимости импортируется для последующего создания на его основе других проектов. Преимущество шаблонного подхода заключается в следующем: процедуры обработки событий сохраняются вместе с шаблоном.

Начните с создания пользовательского диалогового окна, содержащего все элементы управления и настройки, которые необходимо повторно использовать в других проектах. После этого убедитесь, что диалоговое окно выделено. Выберите команду File⇨Export File (или нажмите комбинацию клавиш <Ctrl+E>). После этого на экране появится запрос на ввод имени файла. Затем для создания нового проекта на основе шаблона выполните File⇨Import File, чтобы загрузить ранее сохраненное диалоговое окно.

Список инструкций по созданию диалогового окна

Перед тем как предоставить самостоятельно созданное диалоговое окно на суд зрителей, необходимо удостовериться, что оно работает правильно. Следующий список содержит вопросы, ответы на которые позволяют выяснить наличие и причину потенциальной проблемы.

- Все ли элементы управления одного типа имеют одинаковый размер?
- Равномерно ли распределены элементы управления?

- Не подавляющий ли вид имеет диалоговое окно? Если это так, то следует перегруппировать элементы управления с помощью элемента управления `MultiPage`.
- Ко всем ли элементам управления можно получить доступ с помощью клавиатуры?
- Не повторяются ли комбинации клавиш?
- Правильно ли установлен порядок активизации элементов управления?
- Выполнит ли код VBA необходимые действия, когда пользователь щелкнет на кнопке OK или Отмена?
- Нет ли в тексте ошибок?
- Правильный ли текст заголовка диалогового окна?
- Насколько правильно будет отображаться диалоговое окно при всех разрешениях экрана? Иногда текст, который хорошо выглядит на экране с высоким разрешением, на экране VGA отображается не полностью (выходит за пределы видимой области экрана).
- Обеспечено ли логическое группирование элементов управления (в соответствии с выполняемой функцией)?
- Ограничивают ли элементы управления `ScrollBar` и `SpinButton` диапазон допустимых значений?
- Правильно ли настроены элементы управления `ListBox`?

Глава 14

Примеры пользовательских форм

В ЭТОЙ ГЛАВЕ...

В данной главе приведен ряд полезных и информативных примеров, которые описывают дополнительные методы использования диалоговых окон `UserForm`.

- Применение пользовательских диалоговых окон для создания простого меню
- Выбор диапазона в пользовательском диалоговом окне
- Применение пользовательского диалогового окна в качестве заставки
- Изменение размера пользовательского диалогового окна на экране
- Масштабирование и прокрутка листа с помощью пользовательского диалогового окна
- Описание методик, использующих элемент управления `ListBox`
- Использование элемента управления `MultiPage`

Эти методы можно применять в собственных проектах. Все примеры, приведенные в этой главе, содержаться на прилагаемом к книге компакт-диске.

В главе 15 вы найдете дополнительные примеры использования пользовательских форм

Создание меню с помощью объекта UserForm

Иногда требуется применить пользовательское диалоговое окно в качестве меню. Данный раздел предлагает описание двух способов получения необходимого результата: с помощью элементов управления `CommandButton` и посредством элемента управления `ListBox`.

Использование элементов управления CommandButton

На рис. 14.1 показан пример объекта `UserForm`, в котором использован элемент управления `CommandButton` в качестве простого меню. Создание такого меню не вызывает особых трудностей, а код, обеспечивающий работу пользовательского диалогового окна, предельно прост. Каждый элемент управления `CommandButton` имеет собственную процедуру обработки событий. Например, представленная ниже процедура выполняется при щелчке на кнопке `CommandButton1`.

```
Private Sub CommandButton1_Click()
    Call Macro1
    Unload Me
End Sub
```

Эта процедура приводит к вызову макроса Macro1 и закрывает диалоговое окно UserForm. Другие кнопки используют похожие процедуры обработки событий.

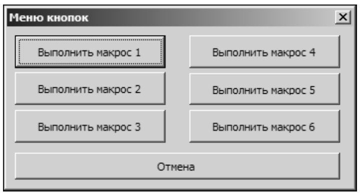


Рис. 14.1. Представленное диалоговое окно содержит элементы управления CommandButton, *предназначенные для создания подобия меню*

Использование элемента управления ListBox

На рис. 14.2 показан еще один пример создания меню, но на этот раз был применен элемент управления ListBox. Перед отображением пользовательского диалогового окна вызывается процедура обработки события Initialize. Процедура, приведенная ниже, использует метод AddItem для добавления шести опций в элемент управления ListBox.

```
Private Sub UserForm_Initialize()
    With ListBox1
        .AddItem "Macro1"
        .AddItem "Macro2"
        .AddItem "Macro3"
        .AddItem "Macro4"
        .AddItem "Macro5"
        .AddItem "Macro6"
    End With
End Sub
```

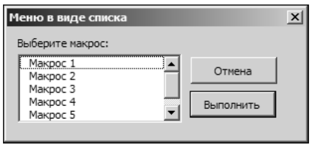


Рис. 14.2. В этом диалоговом окне в качестве меню используется элемент управления ListBox

Кроме того, процедура обработки события добавлена и к кнопке Выполнить для обработки щелчков на ней.

```
Private Sub ExecuteButton_Click()
    Select Case ListBox1.ListIndex
        Case -1
            MsgBox "Выберите макрос"
            Exit Sub
        Case 0: Call Macro1
        Case 1: Call Macro2
        Case 2: Call Macro3
        Case 3: Call Macro4
        Case 4: Call Macro5
        Case 5: Call Macro6
    End Select
    Unload Me
End Sub
```

Данная процедура проверяет значение свойства ListIndex элемента управления ListBox, чтобы определить, какой элемент выбран в списке (если свойство ListIndex равно -1, то не выбран ни один из элементов). После этого запускается соответствующий макрос.

Excel позволяет создавать “настоящие” меню и панели инструментов. Более подробная информация об этом приведена в главах 22–23.

Выбор диапазона

Некоторые встроенные диалоговые окна Excel позволяют пользователю выбирать диапазон. Например, диалоговое окно Поиск решения запрашивает у пользователя два диапазона. Пользователь может или ввести имя диапазона непосредственно, или применить мышь для выделения диапазона на листе.

Пользовательское диалоговое окно также обеспечивает подобную функциональность. Это достигается в результате применения элемента управления `RefEdit`. Данный элемент выглядит иначе, чем элемент выбора диапазона во встроенных диалоговых окнах Excel, однако работает точно так же. Если пользователь щелкнет на небольшой кнопке в правой части элемента управления, то диалоговое окно временно исчезнет, а на экране будет отображен небольшой указатель выбора диапазона (именно так все происходит и при использовании встроенного диалогового окна Excel).

К сожалению, элемент управления `RefEdit` не позволяет использовать при выделении диапазона специальные клавиши. Например, нажатие комбинаций клавиш <Shift+↓> приводит к выделению ячеек до конца столбца.

На рис. 14.3 представлено пользовательское диалоговое окно с добавленным элементом управления `RefEdit`. Это диалоговое окно выполняет простую математическую операцию над всеми не содержащими формул и непустыми ячейками указанного диапазона. Выполняемая операция задается активным переключателем `OptionButton`.

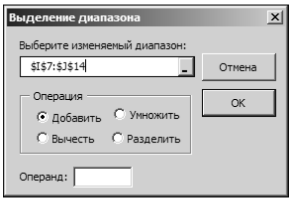


Рис. 14.3. Элемент управления `RefEdit` позволяет выбирать необходимый диапазон

Ниже изложены некоторые соображения, о которых следует помнить при использовании элемента управления `RefEdit`.

- Элемент управления `RefEdit` возвращает текстовую строку, которая представляет выбранный диапазон. Можно преобразовать эту строку в объект `Range`. Для этого используется следующий оператор.

```
Set UserRange = Range(RefEdit1.Text)
```

- Удачной практикой считается инициализация элемента управления `RefEdit` для представления текущего выделения. Для этого в процедуре `UserForm_Initialize` воспользуйтесь таким оператором.

```
RefEdit1.Text = ActiveWindow.RangeSelection.Address
```

- Элемент управления `RefEdit` не всегда возвращает действительный диапазон. Выделение диапазона указателем мыши — это один из способов присвоения значения данному элементу управления. Пользователь может ввести в поле

любой текст, а также отредактировать или удалить уже отображаемый текст. Таким образом, предварительно необходимо убедиться, что диапазон является допустимым. Следующий код — это пример одного из способов проверки допустимости введенного значения. Если определено, что значение неправильное, то пользователю выдается сообщение, а элемент управления RefEdit становится активным, предоставляя возможность ввести корректный диапазон.

```
On Error Resume Next
Set UserRange = Range(RefEdit1.Text)
If Err <> 0 Then
    MsgBox "Неправильный диапазон"
    RefEdit1.SetFocus
    Exit Sub
End If
On Error GoTo 0
```

- Пользователь может щелкнуть на вкладке одного из листов при выборе диапазона, применяя элемент управления RefEdit. Поэтому не всегда выбранный диапазон находится на активном рабочем листе. Если пользователем выбран другой лист, то адрес диапазона указывается после имени листа, на котором этот диапазон находится.

```
Лист2!$A$1:$C:4
```

- Если необходимо получить от пользователя выделение в виде одной ячейки, то можно указывать верхнюю левую ячейку выделенного диапазона. В данном случае воспользуйтесь следующим оператором.

```
Set OneCell = Range(RefEdit1.Text).Range("A1")
```

ПЕРЕКРЕСТНАЯ ССЫЛКА

Как отмечалось в главе 12, можно применить функцию VBA InputBox, чтобы позволить пользователю ввести диапазон.

Создание заставки

Некоторые разработчики предпочитают отображать определенную вступительную информацию при запуске приложения. Эта методика приобрела название *заставки*. Без сомнения, все пользователи видели заставку Excel, которая отображается несколько секунд при запуске программы.

В приложении Excel заставку можно создать с помощью пользовательского диалогового окна. Приведенный ниже пример представлен диалоговым окном, которое отображается автоматически и исчезает по истечении пяти секунд. Следуйте таким инструкциям для создания заставки проекта.

1. Создайте рабочую книгу.
2. Запустите VBE и вставьте новое диалоговое окно UserForm в проект. Код в этом примере предполагает, что объект UserForm называется UserForm1.
3. Поместите любые необходимые элементы управления в только что созданное диалоговое окно UserForm1. Например, вам может понадобиться расположить элемент управления Image, который будет содержать логотип компании. На рис. 14.4 показан пример такого диалогового окна.



Рис. 14.4. Эта заставка отображается при открытии рабочей книги в течение указанного в коде временного интервала

4. Вставьте следующую процедуру в модуль кода для объекта `ThisWorkbook` (`ЭтаКнига`).

```
Private Sub Workbook_Open()
    UserForm1.Show
End Sub
```

5. Вставьте приведенную далее процедуру в модуль кода для объекта `UserForm1` (эта процедура обеспечивает пятисекундную задержку).

```
Private Sub UserForm_Activate()
    Application.OnTime Now + _
      TimeValue("00:00:05"), "KillTheForm"
End Sub
```

6. Вставьте следующую процедуру в общий модуль VBA.

```
Private Sub KillTheForm()
    Unload UserForm1
End Sub
```

При открытии рабочей книги будет выполнена процедура `Workbook_Open`. Она отображает диалоговое окно `UserForm`. В этот момент генерируется событие `Activate`, которое приводит к выполнению процедуры `UserForm_Activate`. Данная процедура использует метод `OnTime` объекта `Application` для выполнения процедуры `KillTheForm` в определенный момент времени. Однако предварительно определена задержка в пять секунд с момента возникновения события `Activate`. Процедура `KillTheForm` просто выгружает диалоговое окно `UserForm` из памяти.

7. В качестве необязательного действия можно добавить элемент управления `CommandButton` с именем `CancelButton`, установить его свойство `Cancel` в значение `True` и добавить представленную ниже процедуру обработки события в модуль кода формы `UserForm`.

```
Private Sub CancelButton_Click()
    KillTheForm
End Sub
```

Таким образом, пользователь сможет закрыть заставку, перед тем как пройдет указанное время задержки. Окно будет закрыто также в результате нажатия клавиши <Esc>. Эту небольшую кнопку можно разместить за другим объектом, чтобы она не была видна.

Помните, что заставка не будет отображаться на экране, пока рабочая книга не полностью загружена. Другими словами, если необходимо отобразить заставку только с той целью, чтобы пользователю было на что смотреть, пока загружается рабочая книга, то описанная выше методика для этого не подходит.

Если вам необходимо выполнить VBA-процедуру при запуске документа, то можете отобразить пользовательскую форму в немодальном режиме так, что код будет продолжать выполняться. Для этого воспользуйтесь процедурой `Workbook_Open`.

```
Private Sub Workbook_Open()
    UserForm1.Show vbModeles
    ' Остальной код
End Sub
```

Отключение кнопки закрытия пользовательского диалогового окна

Если пользовательское диалоговое окно уже отображено на экране, щелчок на кнопке Закрыть (× в правом верхнем углу) приведет к выгрузке формы `UserForm` из памяти. Бывают ситуации, когда этого допускать нельзя. Например, необходимо, чтобы диалоговое окно `UserForm` закрывалась только с помощью щелчка на специальной кнопке `CommandButton`.

Несмотря на то, что реально отключить кнопку Закрыть невозможно, вы вправе предотвратить закрытие диалогового окна, вызванное щелчком на этой кнопке. Для этого воспользуйтесь обработчиком события `QueryClose`.

Следующая процедура, которая расположена в модуле кода диалогового окна `UserForm`, выполняется перед закрытием формы (т.е. в момент возникновения события `QueryClose`).

```
Private Sub UserForm_QueryClose _
  (Cancel As Integer, CloseMode As Integer)
    If CloseMode = vbFormControlMenu Then
        MsgBox "Щелкните на кнопке ОК для закрытия формы"
        Cancel = True
    End If
End Sub
```

Процедура `UserForm_QueryClose` использует два аргумента. Аргумент `CloseMode` содержит значение, которое указывает на причину возникновения события `QueryClose`. Если значение аргумента `CloseMode` равно `vbFormControlMenu` (встроенная константа), то это означает, что пользователь щелкнул на кнопке Закрыть. В таком случае будет отображено сообщение; аргумент `Cancel` устанавливается в значение `True`, и форма не закрывается.

Помните, что пользователь может нажать комбинацию клавиш <Ctrl+Break> и прекратить выполнение макроса. В этом примере нажатие комбинации <Ctrl+Break> во время отображения формы `UserForm` на экране приведет к тому, что пользовательское диалоговое окно будет закрыто. Чтобы этого не произошло, выполните представленный ниже оператор перед отображением диалогового окна `UserForm`.

```
Application.EnableCancelKey = xlDisabled
```

Прежде чем добавлять этот оператор, удостоверьтесь, что приложение не содержит ошибок. В противном случае возникнет ситуация, когда невозможно будет прекратить выполнение бесконечного цикла.

Изменение размера диалогового окна

Многие приложения используют окна, которые могут изменять собственный размер. Например, диалоговое окно Excel Автоформат (которое отображается после выбора команды Формат⇨Автоформат) увеличится в размере после того, как пользователь щелкнет на кнопке Параметры.

Приведенный ниже пример демонстрирует предоставление пользовательскому диалоговому окну возможности динамически изменять свой размер. Изменение размера диалогового окна выполняется с помощью значений свойств `Width` и `Height` объекта `UserForm`.

На рис. 14.5 показан пример диалогового окна в том виде, в котором оно отображается первоначально. На рис. 14.6 отображено диалоговое окно после щелчка на кнопке Параметры. Обратите внимание на то, что надпись на кнопке изменяется в зависимости от размера диалогового окна.

При создании пользовательского диалогового окна определите ему максимальный размер, чтобы получить доступ ко всем элементам управления. После этого воспользуйтесь процедурой `UserForm_Initialize` для установки размеров диалогового окна по умолчанию (меньшие за максимальные).

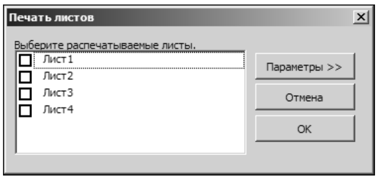


Рис. 14.5. Пример диалогового окна со стандартными размерами

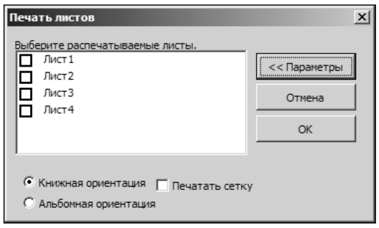


Рис. 14.6. То же самое диалоговое окно с увеличенным размером для отображения дополнительных параметров

Описанный пример предназначен для печати рабочих листов активной книги. Он позволяет пользователю указать листы, которые необходимо распечатать. Ниже приведена процедура обработки события, которая выполняется при щелчке на кнопке Параметры.

```
Private Sub OptionsButton_Click()
    If OptionsButton.Caption = "Параметры >>" Then
        Me.Height = 164
        OptionsButton.Caption = "<< Параметры"
    Else
        Me.Height = 128
        OptionsButton.Caption = "Параметры >>"
    End If
End Sub
```

Эта процедура проверяет значение свойства `Caption` объекта `CommandButton` и устанавливает значение свойства `Height` объекта `UserForm` в соответствии с полученным значением свойства `Caption`.



Если элементы управления не отображены по причине того, что они находятся за пределами границы диалогового окна, комбинации клавиш, соответствующие этим элементам управления, будут продолжать функционировать. В приводимом примере пользователь может нажать комбинацию <Alt+L> (для перехода к альбомной ориентации страницы), даже когда соответствующий элемент управления не виден на экране. Для того чтобы заблокировать доступ к невидимым элементам управления, создайте код, который будет отключать элементы управления, если они скрываются.

Масштабирование и прокрутка листа в пользовательском диалоговом окне

При отображении диалогового окна зачастую необходимо иметь возможность перемещаться по листу для проверки содержимого различных диапазонов. Обычно такая операция невозможна, если диалоговое окно отображается на экране.

> Начиная с Excel 2000, диалоговое окно `UserForm` может становиться немодальным. Это означает, что допускается не закрывать диалоговое окно перед активизацией рабочей книги и выполнением других операций в среде Excel. Метод `Show` объекта `UserForm` по умолчанию определяет модальную форму отображения окна. Для того чтобы отобразить немодальное диалоговое окно, необходимо воспользоваться следующим оператором.
>
> ```
> UserForm1.Show vbModeless
> ```

Пример, приведенный в этом разделе, демонстрирует, как использовать элемент управления `ScrollBar` для прокрутки и масштабирования листа при активном диалоговом окне. На рис. 14.7 отображен пример окна изменения масштаба рабочего листа (в диапазоне от 100% до 400%) с помощью элемента управления `ScrollBar`, находящегося в верхней части диалогового окна. Два элемента управления `ScrollBar` в нижней части диалогового окна позволяют прокручивать лист по горизонтали и по вертикали.

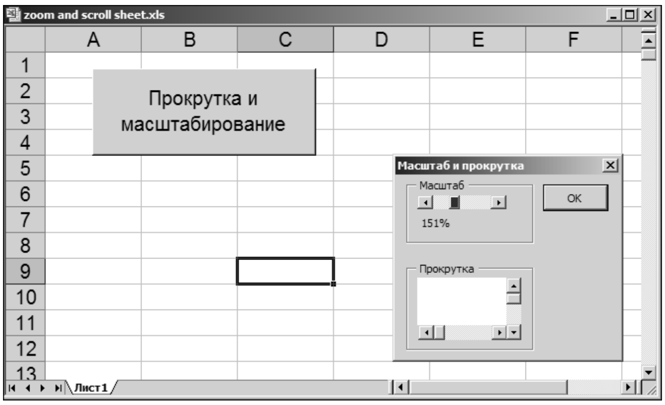


Рис. 14.7. Элементы управления `ScrollBar` позволяют прокручивать лист и изменять его масштаб

Код данного примера достаточно прост.

```
Private Sub UserForm_Initialize()
    LabelZoom.Caption = ActiveWindow.Zoom
'   Масштабирование
    With ScrollBarZoom
        .Min = 10
        .Max = 400
        .SmallChange = 1
        .LargeChange = 10
        .Value = ActiveWindow.Zoom
    End With
```

```
'   Горизонтальная прокрутка
    With ScrollBarColumns
        .Min = 1
        .Max = 256
        .Value = ActiveWindow.ScrollColumn
        .LargeChange = 25
        .SmallChange = 1
    End With

'   Вертикальная прокрутка
    With ScrollBarRows
        .Min = 1
        .Max = ActiveSheet.Rows.Count
        .Value = ActiveWindow.ScrollRow
        .LargeChange = 25
        .SmallChange = 1
    End With
End Sub
```

Эта процедура устанавливает значения различных свойств элементов управления ScrollBar. Значения определяются на основе данных, полученных из активного окна.

При использовании элемента управления ScrollBarZoom выполняется процедура ScrollBarZoom_Change (которая приведена ниже). Эта процедура устанавливает значение свойства Zoom объекта ActiveWindow равным значению свойства Value элемента управления ScrollBar. Кроме того, изменяется текст подписи, которая представляет текущий масштаб рабочего листа.

```
Private Sub ScrollBarZoom_Change()
    With ActiveWindow
        .Zoom = ScrollBarZoom.Value
        LabelZoom = .Zoom & "%"
    End With
End Sub
```

Прокрутка листа осуществляется с помощью двух процедур. Эти процедуры устанавливают значение свойств ScrollRow и ScrollColumns объекта ActiveWindow равными значениям свойств Value элементов управления ScrollBar.

```
Private Sub ScrollBarColumns_Change()
    ActiveWindow.ScrollColumn = ScrollBarColumns.Value
End Sub

Private Sub ScrollBarRows_Change()
    ActiveWindow.ScrollRow = ScrollBarRows.Value
End Sub
```

Вместо использования события Change в предыдущих процедурах, лучше применить событие Scroll. Разница заключается в том, что событие наступает при прокрутке и масштабировании документа.

Использование элемента управления ListBox

Элемент управления ListBox довольно гибкий в использовании, но работа с ним может оказаться достаточно сложной. В данном разделе приведен ряд простых примеров, которые демонстрируют распространенные методики использования элемента управления ListBox.

В большинстве случаев методики, описанные в этом разделе, применяются и к элементу управления ComboBox.

Об элементе управления ListBox

Ниже изложены рекомендации, о которых необходимо помнить при работе с элементом управления `ListBox`. Примеры данного раздела наглядно иллюстрируют эти советы.

- Опции списка элемента управления `ListBox` могут извлекаться из диапазона ячеек (определяемого свойством `RowSource`) или добавляться с помощью VBA (для этого используется метод `AddItem`).
- Элемент управления `ListBox` может быть применен для выделения одной и нескольких опций. Соответствующее поведение определяется значением свойства `MultiSelect`.
- Если элемент управления `ListBox` не настроен на выделение нескольких опций, то значение элемента управления `ListBox` может связываться с ячейкой листа с помощью свойства `ControlSource`.
- Элемент управления `ListBox` может отображаться без предварительно выбранной опции (для этого необходимо установить свойство `ListIndex` в значение -1). Но как только пользователь выделит одну из опций списка, снять выделение будет невозможно.
- Элемент управления `ListBox` может содержать несколько столбцов (что указывается в свойстве `ColumnCount`) и даже описательные заголовки (для этого используется свойство `ColumnHeads`).
- Вертикальный размер элемента управления `ListBox`, помещенного в пользовательское диалоговое окно, не всегда совпадает с вертикальным размером объекта `UserForm` на экране.
- Опции списка элемента управления `ListBox` могут отображаться в виде флажков, если разрешено выделение нескольких элементов, или в виде переключателей, если поддерживается только единичное выделение. Это поведение определяется значением свойства `ListStyle`.

НА ЗАМЕТКУ

Для получения полной информации о свойствах и методах элемента управления `ListBox` обратитесь к диалоговому справочному руководству.

Добавление опций в элемент управления ListBox

Перед отображением пользовательского диалогового окна, которое содержит элемент управления `ListBox`, необходимо заполнить элемент управления `ListBox` необходимыми опциями. Элемент управления `ListBox` заполняется на этапе разработки проекта посредством указания диапазона ячеек, которые содержат необходимые данные. Его также можно заполнить на этапе выполнения, воспользовавшись кодом VBA для добавления всех опций списка.

Примеры настоящего раздела предполагают следующее.

- Используется диалоговое окно `UserForm` с именем `UserForm1`.
- Диалоговое окно `UserForm1` содержит элемент управления `ListBox`, который называется `ListBox1`.
- Рабочая книга содержит лист `Лист1`, в диапазоне `A1:A12` которого определены опции, отображаемые в элементе управления `ListBox`.

ДОБАВЛЕНИЕ ОПЦИЙ В ЭЛЕМЕНТ УПРАВЛЕНИЯ LISTBOX НА ЭТАПЕ РАЗРАБОТКИ

Для того чтобы иметь возможность добавить опции элемента управления `ListBox` на этапе разработки, элементы должны храниться в диапазоне ячеек рабочей книги. Воспользуйтесь свойством `RowSource` для указания диапазона, который содержит опции элемента управления `ListBox`. На рис. 14.8 показано окно Properties для элемента управления `ListBox`. Свойство `RowSource` установлено в значение `Лист1!A1:A12`. Когда диалоговое окно `UserForm` отображается на экране, элемент управления `ListBox` содержит двенадцать опций из этого диапазона. Опции добавляются в элемент управления `ListBox` на этапе разработки, сразу после того, как диапазон определяется в качестве значения свойства `RowSource`.

Удостоверьтесь, что в значении свойства `RowSource` присутствует имя листа. В противном случае элемент управления `ListBox` будет применять указанный диапазон в активном рабочем листе. В некоторых случаях необходимо идентифицировать диапазон максимально точно, указав даже имя рабочей книги. Например: `[Книга1.xls]Лист1!A1:A12`.

ДОБАВЛЕНИЕ ОПЦИЙ В ЭЛЕМЕНТ УПРАВЛЕНИЯ LISTBOX НА ЭТАПЕ ВЫПОЛНЕНИЯ

Чтобы добавить опции элемента управления `ListBox` на этапе выполнения, необходимо реализовать следующее.

- С помощью кода определить значение свойства `RowSource`, чтобы указать диапазон, хранящий необходимые данные.
- Создать код, использующий метод `AddItem` для добавления опций в элемент управления `ListBox`.

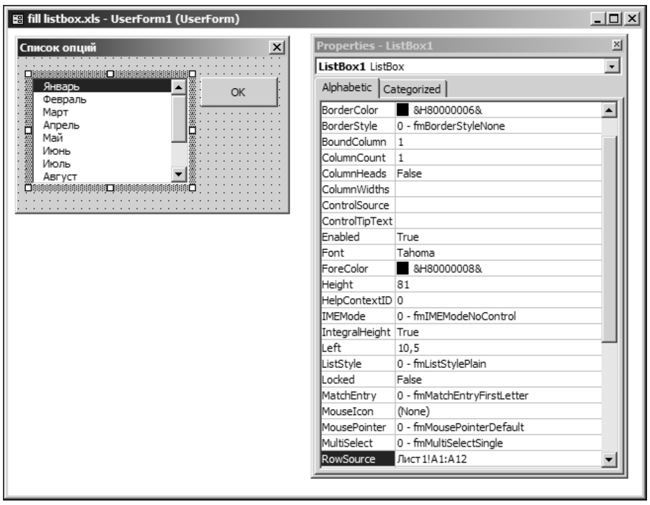


Рис. 14.8. Установка значения свойства `RowSource` на этапе разработки

Значение свойства RowSource устанавливается с помощью кода, а не в окне Properties. Например, представленная далее процедура устанавливает значения свойства RowSource элемента управления ListBox перед тем, как отображается диалоговое окно UserForm. В этом случае опции состоят из значений в ячейках диапазона Categories рабочего листа Budget.

```
UserForm1.ListBox1.RowSource = "Budget!Categories"
UserForm1.Show
```

Если опции элемента управления ListBox не содержатся в диапазоне ячеек листа, то можно создать специальный код VBA для заполнения элемента управления ListBox перед кодом отображения диалогового окна. Следующая процедура заполняет элемент управления ListBox названиями месяцев года с помощью метода AddItem.

```
Sub ShowUserForm2()
'   Заполнение элемента управления ListBox
    With UserForm2.ListBox1
        .RowSource=""
        .AddItem "Январь"
        .AddItem "Февраль"
        .AddItem "Март"
        .AddItem "Апрель"
        .AddItem "Май"
        .AddItem "Июнь"
        .AddItem "Июль"
        .AddItem "Август"
        .AddItem "Сентябрь"
        .AddItem "Октябрь"
        .AddItem "Ноябрь"
        .AddItem "Декабрь"
    End With
    UserForm2.Show
End Sub
```

ВНИМАНИЕ

В предыдущем коде свойство RowSource сначала приравнивалось к пустой строке (чтобы, таким образом, избежать потенциальной ошибки, когда в окне Properties свойство RowSource элемента управления ListBox имеет определенное значение). Если попытаться добавить опции в элемент управления ListBox с ненулевым значением свойства RowSource, то будет выведено сообщение об ошибке ("Permission denied").

Кроме того, можно использовать метод AddItem для получения опций элемента управления ListBox, хранящихся в диапазоне ячеек. Ниже рассмотрен пример заполнения элемента управления ListBox содержимым диапазона A1:A12 листа Лист1.

```
For Row = 1 To 12
  UserForm1.ListBox1.AddItem Sheets("Лист1").Cells(Row, 1)
Next Row
```

Если данные хранятся в одномерном массиве, то можно назначить массив элементу управления с помощью единственной инструкции. Предположим, что у нас есть массив dData, который содержит 50 элементов. Следующий оператор создает список из 50 опций на основе элемента управления ListBox, называющегося ListBox1.

```
ListBox1.List = dData
```

ДОБАВЛЕНИЕ В ЭЛЕМЕНТ УПРАВЛЕНИЯ LISTBOX ТОЛЬКО УНИКАЛЬНЫХ ОПЦИЙ

В определенных случаях возникает необходимость в заполнении элемента управления `ListBox` уникальными (неповторяющимися) опциями из существующего списка. Предположим, у нас есть лист, который содержит данные о потребителях. Один из столбцов может содержать названия стран (рис. 14.9). Необходимо заполнить элемент управления `ListBox` названиями стран, в которых проживают потребители.

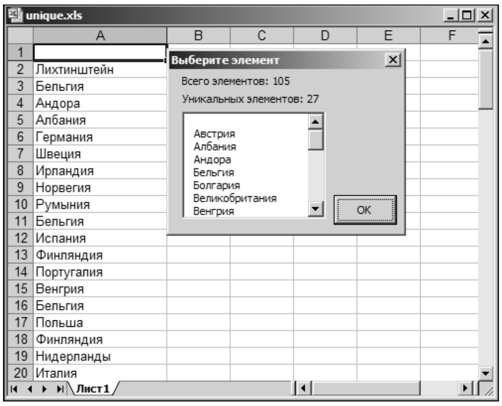


Рис. 14.9. Объект `Collection` используется для заполнения элемента управления `ListBox` уникальными значениями из столбца `A`

Один из методов заполнения предполагает использование объекта `Collection`. Элементы в объект `Collection` добавляются с помощью следующего синтаксиса.

```
object.Add item, key, before, after
```

Аргумент `key` (если он используется) содержит *уникальную* текстовую строку, которая необходима для получения доступа к элементам коллекции. Если к коллекции добавить неуникальный ключ, то возникнет ошибка, в результате элемент добавлен не будет. Этим можно воспользоваться и создать коллекцию, которая содержит только уникальные элементы.

Представленная далее процедура демонстрирует использование этого способа. Процедура начинается с декларации нового объекта `Collection` — `NoDupes`. Предполагается, что диапазон, называющийся `Data`, содержит список элементов, часть которых повторяется.

В коде циклически просматриваются ячейки диапазона, и в коллекцию `NoDupes` добавляются значения только уникальных ячеек. Кроме того, значение ячейки (преобразованное в строку) используется в качестве значения аргумента `key`. Применение оператора `On Error Resume Next` приводит к тому, что в коде VBA игнорируется ошибка, которая возникает при добавлении в коллекцию неуникального ключа. Если возникает ошибка, элемент в коллекцию не добавляется — это именно то поведение, которого требуется добиться. Затем процедура передает элементы коллекции `NoDupes` в элемент управления `ListBox`. Диалоговое окно `UserForm` также содержит подпись, которая указывает количество уникальных элементов коллекции.

```
Sub RemoveDuplicates1()
    Dim AllCells As Range, Cell As Range
    Dim NoDupes As New Collection

    On Error Resume Next
    For Each Cell In Range("Data")
        NoDupes.Add Cell.Value, CStr(Cell.Value)
    Next Cell
    On Error GoTo 0

'   Добавить уникальные элементы в элемент управления ListBox
    For Each Item In NoDupes
        UserForm1.ListBox1.AddItem Item
    Next Item

'   Отобразить количество
    UserForm1.Label1.Caption = _
      "Уникальные элементы: " & NoDupes.Count

'   Отобразить диалоговое окно UserForm
    UserForm1.Show
End Sub
```

Несколько отличная версия этой процедуры представлена на прилагаемом к книге компакт-диске.

Определение выделенной опции

Примеры предыдущих разделов отображали диалоговое окно `UserForm` с элементом управления `ListBox`, который содержит список из нескольких опций. Эти процедуры не включают главной функциональности: определения опции или опций, которые выбраны пользователем.

Далее будет использован элемент управления `ListBox` с выделением одной опции списка — его свойство `MultiSelect` должно иметь значение 0.

Чтобы определить, какая опция выбрана, необходимо узнать значение свойства `Value` элемента управления `ListBox`. Оператор, показанный ниже, отображает текст выделенной в объекте `ListBox1` опции.

```
MsgBox ListBox1.Value
```

Если не выделена ни одна опция, то выполнение оператора приведет к возникновению ошибки.

Чтобы узнать расположение выделенной опции в списке (а не только его содержимое), воспользуйтесь значением свойства `ListIndex` элемента управления `ListBox`. Следующий пример демонстрирует простое окно сообщения, в котором указан номер выделенной опции элемента управления `ListBox`.

```
MsgBox "Вы выбрали опцию #" & ListBox1.ListIndex
```

Если не выделена ни одна опция, то свойство `ListIndex` будет иметь значение -1.

Нумерация опций в элементе управления `ListBox` начинается с 0, а не с 1. Таким образом, значение свойства `ListIndex` для первого элемента равно 0. Для последнего элемента списка значение свойства равно значению свойства `ListCount-1`.

Определение нескольких выделенных опций

Как правило, свойство `MultiSelect` элемента управления `ListBox` имеет значение 0; таким образом, пользователь может выбрать только одну опцию в элементе управления `ListBox`.

Если элемент управления `ListBox` позволяет выделять несколько опций (т.е. его свойство `MultiSelect` установлено в значение 1 или 2), то при попытке доступа к значениям свойств `ListIndex` или `Value` элемента управления `ListBox` возникнет ошибка. Вместо указанных свойств необходимо использовать свойство `Selected`, возвращающее массив, первый элемент которого имеет индекс 0. Например, представленный ниже оператор отображает значение `True`, если выделена первая опция элемента управления `ListBox`.

```
MsgBox ListBox1.Selected(0)
```

НА КОМПАКТ-ДИСКЕ

На прилагаемом к книге компакт-диске содержится рабочая книга, в которой отображены принципы идентификации выделенных опций списка в элементе управления `ListBox`. Этот способ работает при выделении и одной, и нескольких опций.

Следующий код взят из примера рабочей книги, которая расположена на компакт-диске. В данном коде просматриваются все опции списка элемента управления `ListBox`. Если опция выделена, то текст опции добавляется к переменной `Msg`. В конце все выделенные опции отображаются в окне сообщения.

```
Private Sub OKButton_Click()
    Msg = ""
    For i = 0 To ListBox1.ListCount - 1
        If ListBox1.Selected(i) Then _
          Msg = Msg & ListBox1.List(i) & vbCrLf
    Next i
    MsgBox "Вы выбрали: " & vbCrLf & Msg
    Unload Me
End Sub
```

На рис. 14.10 показан результат выполнения этой процедуры при выделенных нескольких опциях в элементе управления `ListBox`.

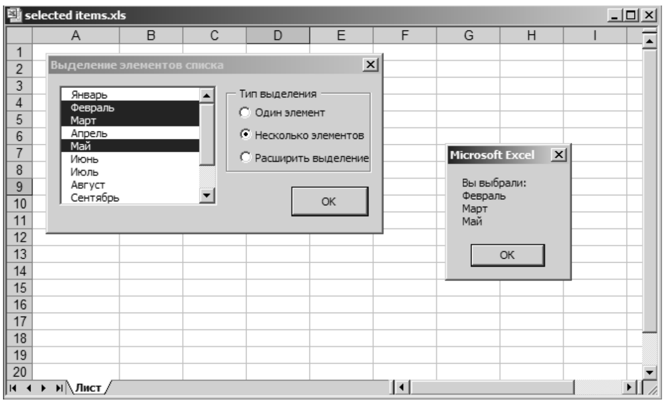


Рис. 14.10. Это окно сообщения содержит список опций, которые выделены в элементе управления `ListBox`

Несколько списков в одном элементе управления ListBox

Приведенный ниже пример демонстрирует создание элемента управления `ListBox`, изменяющего свое содержимое в зависимости от того, какие переключатели `OptionButton` установил пользователь.

На рис. 14.11 показан пример диалогового окна `UserForm`. Элемент управления `ListBox` получает список значений из диапазона на листе. Процедуры, обрабатывающие событие `Click` для элементов управления `OptionButton`, устанавливают значение свойства `RowSource` элемента управления `ListBox` равным необходимому диапазону. Одна из таких процедур представлена далее.

```
Private Sub obMonths_Click()
    ListBox1.RowSource = "Лист1!Месяцы"
End Sub
```

Щелчок на элементе управления `OptionButton`, называющемся `obMonths`, приводит к изменению значения свойства `RowSource` элемента управления `ListBox`, что заставляет его использовать диапазон `Месяцы` на листе `Лист1`.

Передача опций элемента управления ListBox

В некоторых приложениях требуется выбрать несколько элементов списка. Зачастую следует создать новый список на основе выделенных элементов. Примером такой ситуации служит диалоговое окно Управление панелями инструментов, которое возникает при щелчке на кнопке Вложить в диалоговом окне Настройка (это диалоговое окно можно отобразить, выбрав Вид⇨Панели инструментов⇨Настройка).

На рис. 14.12 показано диалоговое окно с двумя элементами управления `ListBox`. Кнопка Добавить добавляет элемент, выделенный в левом элементе управления `ListBox`, в правый элемент управления `ListBox`. Кнопка Удалить удаляет выделенный элемент из правого списка. Флажок определяет поведение при добавлении в список повторяющихся элементов. Если флажок Разрешить дублирование не установлен, то в случае, если пользователь попытается добавить элемент, который уже присутствует в списке, будет отображено окно сообщения.

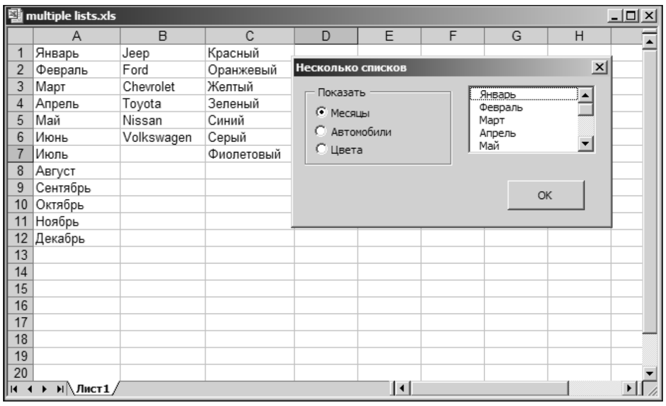


Рис. 14.11. Содержимое элемента управления `ListBox` зависит от того, какой из элементов управления `OptionButton` выбран в текущий момент

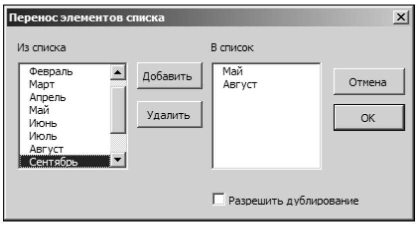


Рис. 14.12. Создание списка на основе другого списка

Код этого примера на удивление прост. Ниже приведена процедура, которая выполняется по щелчку на кнопке Добавить.

```
Private Sub AddButton_Click()
    If ListBox1.ListIndex = -1 Then Exit Sub
    If Not cbDuplicates Then
'       Проверить существование элемента
        For i = 0 To ListBox2.ListCount - 1
            If ListBox1.Value = ListBox2.List(i) Then
                Beep
                Exit Sub
            End If
        Next i
    End If
    ListBox2.AddItem ListBox1.Value
End Sub
```

Код для управления кнопкой Удалить еще проще.

```
Private Sub DeleteButton_Click()
    If ListBox2.ListIndex = -1 Then Exit Sub
    ListBox2.RemoveItem ListBox2.ListIndex
End Sub
```

Обратите внимание, что обе процедуры проверяют существование выделенного элемента. Если значение свойства `ListIndex` элемента управления `ListBox` равно -1, значит, не выделен ни один элемент. В результате процедура завершается.

Перемещение опции в списке элемента управления ListBox

В данном разделе вы узнаете, как можно перемещать опции вверх и вниз в списке элемента управления `ListBox`. В VBA подобный метод применяется для предоставления пользователю возможности указать порядок просмотра элементов управления в диалоговом окне `UserForm`.

На рис. 14.13 показано диалоговое окно, которое содержит элемент управления `ListBox` и два элемента управления `CommandButton`. Щелчок на кнопке Вверх приведет к перемещению выделенной опции вверх по списку элемента управления `ListBox`. Щелчок на кнопке Вниз приведет к перемещению выделенной опции вниз по списку.

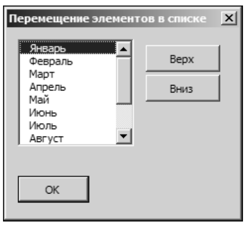


Рис. 14.13. Кнопки позволяют пользователю перемещать опции списка в элементе управления `ListBox`

Процедуры обработки событий двух элементов управления CommandButton выглядят следующим образом.

```
Private Sub MoveUpButton_Click()
    If ListBox1.ListIndex <= 0 Then Exit Sub
    NumItems = ListBox1.ListCount
    Dim TempList()
    ReDim TempList(0 To NumItems - 1)
'   Заполнить массив опциями списка
    For i = 0 To NumItems - 1
        TempList(i) = ListBox1.List(i)
    Next i
'   Выделенные опции
    ItemNum = ListBox1.ListIndex
'   Обменять элементы
    TempItem = TempList(ItemNum)
    TempList(ItemNum) = TempList(ItemNum - 1)
    TempList(ItemNum - 1) = TempItem
    ListBox1.List = TempList
'   Изменить индекс списка
    ListBox1.ListIndex = ItemNum - 1
End Sub

Private Sub MoveDownButton_Click()
    If ListBox1.ListIndex = ListBox1.ListCount - 1 Then Exit Sub
    NumItems = ListBox1.ListCount
    Dim TempList()
    ReDim TempList(0 To NumItems - 1)
'   Заполнить массив опциями списка
    For i = 0 To NumItems - 1
        TempList(i) = ListBox1.List(i)
    Next i
'   Выделенные опции
    ItemNum = ListBox1.ListIndex
'   Обменять элементы
    TempItem = TempList(ItemNum)
    TempList(ItemNum) = TempList(ItemNum + 1)
    TempList(ItemNum + 1) = TempItem
    ListBox1.List = TempList
'   Изменить индекс списка
    ListBox1.ListIndex = ItemNum + 1
End Sub
```

Эти процедуры выполняются достаточно неплохо, но может оказаться, что, по неизвестной причине, достаточно быстрые щелчки на кнопке не приводят к требуемому результату. Например, можно трижды быстро щелкнуть на кнопке Вниз, но элемент списка переместится только на две позиции вниз. Решением этой проблемы является добавление обработчика события DblClick для каждого элемента управления CommandButton. Процедуры, которые всего лишь вызывают процедуры обработки события Click, выглядят следующим образом.

```
Private Sub MoveUpButton_DblClick _
  (ByVal Cancel As MSForms.ReturnBoolean)
    Call MoveUpButton_Click
End Sub

Private Sub MoveDownButton_DblClick _
  (ByVal Cancel As MSForms.ReturnBoolean)
    Call MoveDownButton_Click
End Sub
```

Работа с элементами управления ListBox, содержащими несколько столбцов

Как правило, элемент управления `ListBox` содержит один столбец, в котором отображается один список. Однако можно создать элемент управления `ListBox`, который содержит несколько столбцов и порой даже несколько столбцов с заголовками. На рис. 14.14 отображен элемент управления `ListBox` с несколькими столбцами, который получает данные из диапазона ячеек рабочего листа.

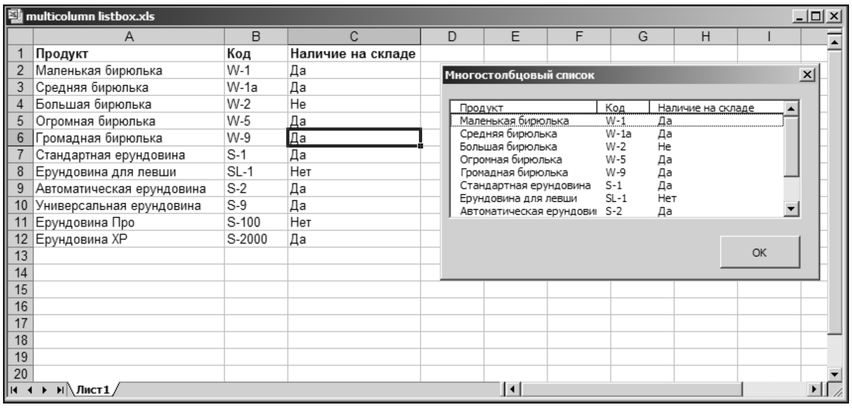


Рис. 14.14. Этот элемент управления `ListBox` *отображает данные в виде списка из трех столбцов, для каждого из которых указан заголовок*

Для того чтобы создать элемент управления `ListBox` с несколькими столбцами, в которые заносятся данные, хранимые в диапазоне ячеек листа, выполните следующее.

1. Удостоверьтесь, что свойство `ColumnCount` элемента управления `ListBox` установлено в правильное значение, которое соответствует количеству столбцов в элементе управления.
2. Укажите правильный исходный диапазон данных из нескольких столбцов, присвоив соответствующее значение свойству `RowSource` элемента управления `ListBox`.
3. Если необходимо отобразить заголовки столбцов, как это показано на рис. 14.14, установите свойство `ColumnHeads` в значение `True`. Не включайте заголовки столбцов в диапазон рабочего листа, указанный в свойстве `RowSource`. VBA автоматически использует для них строку, которая находится сразу над строкой, указанной в значении свойства `RowSource`.
4. Измените ширину столбцов, присвоив свойству `ColumnWidths` значения, которые указываются в пунктах (1/72 часть дюйма) и разделены точками с запятой. Например, следующее значение свойства `ColumnWidths` определяет ширину трех столбцов списка элемента управления `ListBox`.

```
100;40;30
```

5. Укажите столбец в качестве значения свойства `BoundColumn`. Это свойство определяет столбец, на который указывает ссылка при обращении к свойству `Value` элемента управления `ListBox`.

Чтобы заполнить элемент управления ListBox данными из нескольких столбцов без использования диапазона, необходимо создать двухмерный массив, а затем присвоить массив свойству List элемента управления ListBox. Следующий оператор отображает названия месяцев в столбце 1, а количество дней — в столбце 2 (рис. 14.15). Обратите внимание, что процедура устанавливает свойство ColumnCount в значение 2.

```
Private Sub UserForm_Initialize()
'   Заполнить элемент управления ListBox
    Dim Data(1 To 12, 1 To 2)
    For i = 1 To 12
        Data(i, 1) = Format(DateSerial(2001, i, 1), "mmmm")
    Next i
    For i = 1 To 12
        Data(i, 2) = Day(DateSerial(2001, i + 1, 1) - 1)
    Next i
    ListBox1.ColumnCount = 2
    ListBox1.List = Data
End Sub
```

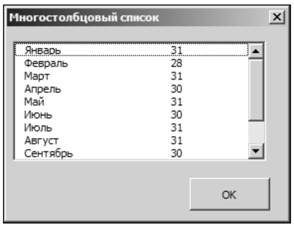


Рис. 14.15. Заполненный данными двухмерного массива элемент управления ListBox с двумя столбцами

Не существует способа определить заголовки столбцов в свойстве ColumnHeads, когда в качестве источника списка применяется массив VBA.

Использование элемента управления ListBox для выделения строк на листе

Пример, приведенный в этом разделе, поможет вам в решении различных практических задач. Он позволяет отображать элемент управления ListBox, который состоит из элементов заполненного диапазона на текущем листе (рис. 14.16). Пользователь может выбрать несколько опций списка в элементе управления ListBox. Щелкнув на кнопке Все, вы выберете все опции, а щелкнув на кнопке Сброс — вызовете отмену выбора всех опций. Щелчок на кнопке OK приводит к выделению строк, которые соответствуют выделенным опциям элемента управления ListBox. Конечно, можно выделить несколько несмежных диапазонов непосредственно на листе. Эта задача выполняется с помощью клавиши <Ctrl>. Но со временем становиться понятно, что метод, предложенный в этом разделе, намного удобнее.

Выбор нескольких элементов возможен благодаря тому, что свойство MultiSelect элемента управления ListBox установлено в значение 1 - fmMultiSelectMulti. Напротив каждого пункта отображаются “флажки”, так как свойство ListStyle элемента управления ListBox установлено в значение 1 - fmListStyleOption.



Рис. 14.16. Этот элемент управления `ListBox` упрощает выделение строк на рабочем листе

Ниже приведена процедура объекта `UserForm` для обработки события `Initialize`. Эта процедура создает объект `rng`, который состоит из используемого диапазона активного листа. Дополнительный код устанавливает свойства `RowSource` и `ColumnCount` элемента управления `ListBox`, а также изменяет значение свойства `ColumnWidths`, чтобы столбцы элемента управления `ListBox` по ширине соответствовали столбцам активного рабочего листа.

```
Private Sub UserForm_Initialize()
    ColCnt = ActiveSheet.UsedRange.Columns.Count
    Set rng = ActiveSheet.UsedRange
    With ListBox1
        .ColumnCount = ColCnt
        .RowSource = rng.Address
        cw = ""
        For c = 1 To .ColumnCount
            cw = cw & rng.Columns(c).Width & ";"
        Next c
        .ColumnWidths = cw
        .ListIndex = 0
    End With
End Sub
```

Кнопки Все и Сброс (называющиеся `SelectAllButton` и `SelectNoneButton`) имеют простые процедуры обработки событий, которые показаны ниже.

```
Private Sub SelectAllButton_Click()
    For r = 0 To ListBox1.ListCount - 1
        ListBox1.Selected(r) = True
    Next r
End Sub

Private Sub SelectNoneButton_Click()
    For r = 0 To ListBox1.ListCount - 1
        ListBox1.Selected(r) = False
    Next r
End Sub
```

Далее приведена процедура обработки события `OKButton_Click`. Эта процедура создает объект `Range`, называющийся `RowRange`. Он состоит из строк, соответствующих выделенным опциям в элементе управления `ListBox`. Для того чтобы определить факт выделения опции, в коде проверяется значение свойства `Selected` элемента управления `ListBox`. Обратите внимание, что используется функция `Union` для добавления дополнительных диапазонов к объекту `RowRange`.

```
Private Sub OKButton_Click()
    Dim RowRange As Range
    RowCnt = 0
    For r = 0 To ListBox1.ListCount - 1
        If ListBox1.Selected(r) Then
            RowCnt = RowCnt + 1
            If RowCnt = 1 Then
                Set RowRange = ActiveSheet.Rows(r + 1)
            Else
                Set RowRange = _
                  Union(RowRange, ActiveSheet.Rows(r + 1))
            End If
        End If
    Next r
    If Not RowRange Is Nothing Then RowRange.Select
    Unload Me
End Sub
```

Использование элемента управления ListBox для активизации листа

Пример, приведенный в этой главе, и полезен, и познавателен. В нем использован элемент управления `ListBox` с несколькими столбцами для отображения списка рабочих листов активной рабочей книги. Столбцы содержат следующие данные.

- Имя листа.
- Тип листа (рабочий лист, диаграмма или диалоговый лист Excel 5/95).
- Количество непустых ячеек в листе.
- Состояние листа.

На рис. 14.17 показан пример такого диалогового окна.

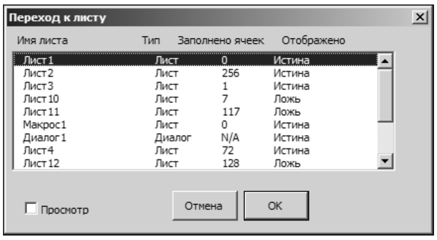


Рис. 14.17. Это диалоговое окно позволяет пользователю активизировать выбранный лист

Код процедуры `UserForm_Initialize` (который приведен ниже) создает двухмерный массив и собирает информацию, циклически просматривая листы активной рабочей книги. После этого массив передается в элемент управления `ListBox`.

```
Private Sub UserForm_Initialize()
    Dim SheetData() As String
    Set OriginalSheet = ActiveSheet
    ShtCnt = ActiveWorkbook.Sheets.Count
    ReDim SheetData(1 To ShtCnt, 1 To 4)
    ShtNum = 1
    For Each Sht In ActiveWorkbook.Sheets
        If Sht.Name = ActiveSheet.Name Then _
          ListPos = ShtNum - 1
        SheetData(ShtNum, 1) = Sht.Name
```

```
        Select Case TypeName(Sht)
            Case "Worksheet"
                SheetData(ShtNum, 2) = "Лист"
                SheetData(ShtNum, 3) = _
                  Application.CountA(Sht.Cells)
            Case "Chart"
                SheetData(ShtNum, 2) = "Диаграмма"
                SheetData(ShtNum, 3) = "N/A"
            Case "DialogSheet"
                SheetData(ShtNum, 2) = "Диалог"
                SheetData(ShtNum, 3) = "N/A"
        End Select
        If Sht.Visible Then
            SheetData(ShtNum, 4) = "Истина"
        Else
            SheetData(ShtNum, 4) = "Ложь"
        End If
        ShtNum = ShtNum + 1
    Next Sht
    With ListBox1
        .ColumnWidths = "100 pt;50 pt;40 pt;50 pt"
        .List = SheetData
        .ListIndex = ListPos
    End With
End Sub
```

Ниже приведена процедура `ListBox1_Click`.

```
Private Sub ListBox1_Click()
    If cbPreview Then _
        Sheets(ListBox1.Value).Activate
End Sub
```

Значение элемента управления `CheckBox` (с названием `cbPreview`) определяет необходимость предварительного просмотра листа после того, как пользователь щелкнет на соответствующей опции списка элемента управления `ListBox`.

Щелчок на кнопке OK (объект `OKButton`) приводит к выполнению процедуры `OKButton_Click`, которая отображена далее.

```
Private Sub OKButton_Click()
    Dim UserSheet As Object
    Set UserSheet = Sheets(ListBox1.Value)
    If UserSheet.Visible Then
        UserSheet.Activate
    Else
        If MsgBox("Отобразить листы?", _
          vbQuestion + vbYesNoCancel) = vbYes Then
            UserSheet.Visible = True
            UserSheet.Activate
        Else
            OriginalSheet.Activate
        End If
    End If
    Unload Me
End Sub
```

Процедура `OKButton_Click` создает переменную объекта, которая представляет выделенный лист. Если лист отображается, то он активизируется. Если лист скрыт, то на экран выводится сообщение, в котором предлагается сделать лист видимым. Если пользователь даст утвердительный ответ на запрос, то лист будет отображен и активизируется. В противном случае активизируется исходный лист (который хранится в переменной `OriginalSheet`).

Двойной щелчок на опции списка в элементе управления ListBox приводит к тому же результату, что и щелчок на кнопке OK. Процедура ListBox1_DblClick, которая отображена ниже, вызывает процедуру OKButton_Click.

```
Private Sub ListBox1_DblClick(ByVal Cancel As _
                        MSForms.ReturnBoolean)
    Call OKButton_Click
End Sub
```

Применение элемента управления MultiPage

Элемент управления MultiPage используется в тех диалоговых окнах UserForm, которые содержат большое количество элементов управления. Он позволяет группировать элементы управления и размещать каждую такую группу на отдельной вкладке.

Элемент управления MultiPage очень гибкий, он предоставляет возможность управления внешним видом и поведением окна. Этот элемент управления включает свойство TabOrientation, принимающее одно из четырех значений.

Окно Toolbox содержит элемент управления, называющийся TabStrip. Элемент управления MultiPage является более гибким, поэтому вам вряд ли придется использовать элемент управления TabStrip.

Применение элемента управления MultiPage может вызвать определенные проблемы. Ниже описаны факторы, которые необходимо учитывать при использовании элемента управления MultiPage.

- Вкладка (или страница), которая отображается поверх всех остальных, определяется значением свойства Value элемента управления MultiPage. Значение 0 соответствует первой вкладке. Значение 1 вызывает отображение второй вкладки и т.д.
- По умолчанию элемент управления MultiPage состоит из двух страниц. Для того чтобы добавить дополнительные вкладки, щелкните на любой вкладке правой кнопкой мыши и выберите New Page из появившегося контекстного меню.
- При работе с элементом управления MultiPage щелкните на вкладке, чтобы установить свойства страницы. Окно Properties будет отображать свойства, значения которых можно изменить.
- Порой сложно выделить сам элемент управления MultiPage, так как щелчок на этом элементе приводит к выделению страницы элемента управления. Для того чтобы выделить только элемент управления, щелкните на его границе. Кроме того, можно воспользоваться клавишей <Tab> для циклического перемещения между элементами управления. Еще одним вариантом выделения элемента управления является выбор MultiPage из раскрывающегося списка окна Properties.
- Если элемент управления MultiPage содержит много вкладок, то установите его свойство MultiRow в значение True, чтобы отобразить вкладки в несколько строк.
- Если необходимо, то вместо вкладок можно отображать кнопки. Достаточно изменить значение свойства Style на 1. Если значение свойства Style равно 2, элемент управления MultiPage не будет отображать ни вкладки, ни кнопки.
- Свойство TabOrientation определяет расположение вкладок на элементе управления MultiPage.

- Для каждой страницы можно установить эффект перехода. Для этого воспользуйтесь свойством `TransitionEffect`. Например, щелчок на вкладке приведет к тому, что новая страница “отодвинет” старую. Применяйте свойство `TransitionPeriod` с целью установить скорость эффекта перехода.

ПЕРЕКРЕСТНАЯ ССЫЛКА

В следующей главе приведены примеры, в которых используется элемент управления `MultiPage`.

Глава 15

Использование диалоговых окон UserForm

В ЭТОЙ ГЛАВЕ...

В этой главе рассмотрены примеры, которые не вошли в главу 13.

- Отображение индикатора текущего состояния.
- Создание “мастера” — интерактивной последовательности диалоговых окон.
- Создание функции, которая эмулирует функцию `MsgBox` в VBA.
- Использование немодальных диалоговых окон `UserForm`.
- Управление несколькими объектами с помощью единственной процедуры обработки события.
- Использование диалогового окна для выбора цвета.
- Отображение информации о ячейке в диалоговом окне `UserForm`.
- Отображение диаграммы в диалоговом окне `UserForm` (два метода).
- Отображение всего листа в диалоговом окне `UserForm`.
- Использование расширенного диалогового окна формы данных.

Приведенные ниже проекты имеют широкое применение; все они основываются на конкретных задачах.

Отображение индикатора текущего состояния

Одним из самых волнующих вопросов для разработчиков приложений Excel является возможность использования индикатора текущего состояния. *Индикатор текущего состояния* — это графический “измеритель”, который отображает текущее состояние выполняемой задачи, например, долго работающего макроса.

До Excel версии 97 создание индикатора текущего состояния было сложной задачей. Теперь все намного проще. В настоящем разделе рассматриваются методы создания индикаторов текущего состояния.

- Макрос, который запускается за пределами диалогового окна `UserForm` (отдельный индикатор текущего состояния).
- Макрос, который запускается из диалогового окна `UserForm`. В этом случае диалоговое окно `UserForm` использует элемент управления `MultiPage` для отображения индикатора текущего состояния, пока выполняется другой макрос.

♦ Макрос, который запускается из диалогового окна `UserForm`. В этом случае диалоговое окно `UserForm` увеличивает свою высоту, а индикатор текущего состояния отображается в нижней части окна.

Используя индикатор текущего состояния, необходимо знать, насколько завершено текущее задание. Способы получения этой информации различаются в зависимости от типа выполняемого макроса. Например, если макрос записывает данные в ячейки (и количество таких ячеек известно), то остается создать код, который будет подсчитывать процентное отношение количества ячеек, содержащих данные.

Отображение текущего состояния в строке состояния окна

Распространенным способом, отображающим состояние выполнения макроса, является использование строки состояния Excel. К недостаткам этого способа можно отнести то, что большинство пользователей не привыкли отслеживать информацию, появляющуюся в строке состояния. Обычно пользователи предпочитают просматривать информацию в специальных окнах.

Для того чтобы отобразить текст в строке состояния Excel, воспользуйтесь следующим оператором.

```
Application.StatusBar = "Пожалуйста, подождите..."
```

Вы вправе обновлять строку состояния в процессе работы макроса. Например, если в макросе используется переменная `Pct`, которая представляет состояние задачи, то можно создать код, который будет периодически выполнять представленный далее оператор.

```
Application.StatusBar = "Выполнение... " & Pct & "% завершено"
```

После того как макрос будет выполнен, верните строку состояния к первоначальному виду. Для этого воспользуйтесь таким оператором.

```
Application.StatusBar = False
```

ВНИМАНИЕ

Помните, что индикатор текущего состояния замедляет работу макроса, так как обновление индикатора требует дополнительного использования ресурсов компьютера. Если главный критерий — быстродействие макроса, то от использования индикатора текущего состояния придется отказаться.

Создание отдельного индикатора текущего состояния

В данном разделе описывается метод использования диалогового окна `UserForm` для отображения текущего состояния задачи, выполняемой макросом.

НА КОМПАКТ-ДИСКЕ

Этот пример находится на прилагаемом к книге компакт-диске.

СОЗДАНИЕ ДИАЛОГОВОГО ОКНА USERFORM

Ниже приведены инструкции по созданию диалогового окна `UserForm`, которое будет отображать текущее состояния выполняемой задачи.

1. Добавьте новое диалоговое окно `UserForm` и измените значение свойства `Caption` на `Состояние`.
2. Добавьте элемент управления `Frame` и назовите его `FrameProgress`.

3. Внутри элемента управления `Frame` создайте элемент управления `Label` и назовите его `LabelProgress`. Удалите заголовок этого элемента управления и сделайте фон элемента управления красным (с помощью свойства `BackColor`). На данный момент размер и расположение этого элемента управления не важны.
4. Добавьте еще один элемент управления `Label` над элементом, чтобы иметь возможность описывать происходящее.
5. Измените размер и расположение элементов управления и диалогового окна `UserForm` — они должны выглядеть, как показано на рис. 15.1.

Вы можете использовать и другой тип форматирования для изменения внешнего вида элементов управления: например, изменить значение свойства `SpecialEffect` для элемента управления `Frame`, как показано на рис. 15.1.

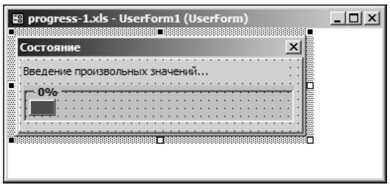


Рис. 15.1. Это диалоговое окно `UserForm` служит контейнером индикатора текущего состояния

СОЗДАНИЕ ПРОЦЕДУРЫ ОБРАБОТКИ СОБЫТИЯ

В данном случае важно, чтобы процедура автоматически запускалась при отображении диалогового окна `UserForm`. Один из вариантов подразумевает использование события `Initialize`. Но это событие возникает еще до того, как диалоговое окно отображается на экране, поэтому такой вариант не подходит. С другой стороны, событие `Activate` возникает в тот момент, когда диалоговое окно `UserForm` отображается на экране, поэтому в данном случае можно остановиться на его использовании.

Вставьте приведенную ниже процедуру в модуль кода диалогового окна `UserForm`. Эта процедура всего лишь вызывает процедуру `Main`, когда диалоговое окно `UserForm` отображается на экране. Процедура `Main`, которая хранится в модуле кода VBA, является фактическим макросом, который будет работать, пока на экране отображается индикатор текущего состояния.

```
Private Sub UserForm_Activate()
    Call Main
End Sub
```

Ниже приведена процедура `Main`. Эта демонстрационная программа просто вставляет случайные числа в ячейки активного листа. В процессе своей работы макрос изменяет ширину элемента управления `Label` и отображает процент завершения задачи в строке заголовка элемента управления `Frame`. Данная процедура лишь иллюстрирует использование индикатора текущего состояния. Но вы всегда можете использовать собственную процедуру, которая выполняет более полезную функцию.

```
Sub Main()
'   Вставить случайные числа в ячейки активного листа
    Cells.Clear
    Counter = 1
    RowMax = 200
    ColMax = 25
    For r = 1 To RowMax
        For c = 1 To ColMax
            Cells(r, c) = Int(Rnd * 1000)
            Counter = Counter + 1
```

```
        Next c
        PctDone = Counter / (RowMax * ColMax)
        Call UpdateProgress(PctDone)
    Next r
    Unload UserForm1
End Sub
```

Процедура `Main` содержит цикл (фактически, два цикла). Внутри цикла вызывается процедура `UpdateProgress`. Она принимает один аргумент: значение в диапазоне от 0 до 100, которое представляет процент завершения выполняемой задачи.

```
Sub UpdateProgress(Pct)
    With UserForm1
      .FrameProgress.Caption = Format(Pct, "0%")
      .LabelProgress.Width = Pct * (.FrameProgress.Width - 10)
      .Repaint
    End With
End Sub
```

СОЗДАНИЕ ПРОЦЕДУРЫ ЗАПУСКА

Все, что осталось сделать, — это написать процедуру отображения диалогового окна `UserForm`. Введите следующий код в модуль VBA.

```
Sub ShowDialog()
    UserForm1.LabelProgress.Width = 0
    UserForm1.Show
End Sub
```

КАК ОНА РАБОТАЕТ

При выполнении процедуры `ShowDialog` ширина объекта `Label` устанавливается в значение 0. После этого вызывается метод `Show` объекта `UserForm1`, что приводит к отображению диалогового окна `UserForm` (которое выполняет роль индикатора текущего состояния). Когда диалоговое окно `UserForm` отображается на экране, происходит событие `Activate`, которое приводит к выполнению процедуры `Main`. Процедура `Main` периодически обновляет ширину элемента управления `Label`. Обратите внимание на то, что процедура использует метод `Repaint` объекта `UserForm`. Без этого оператора изменения в элементе управления `Label` не будут представлены на экране. Перед завершением выполнения процедуры последний оператор выгружает диалоговое окно `UserForm` из памяти.

Для того чтобы модифицировать эту методику, необходимо разобраться с тем, как определяется процент завершения выполняемой задачи. После этого значение состояния задачи можно будет присваивать переменной `PctDone`.

Вывод информации о текущем состоянии с помощью элемента управления MultiPage

В предыдущем примере макрос запускался не из диалогового окна `UserForm`. Если долго выполняющийся макрос необходимо запустить из диалогового окна `UserForm`, то способ, описанный в этом разделе, является более удачным. Итак, сделаем некоторые допущения.

- Проект завершен и отлажен.
- В проекте используется диалоговое окно `UserForm` (без элемента управления `MultiPage`) для запуска долго выполняющегося макроса.
- Существует метод оценки степени завершения выполняемой задачи.

На прилагаемом к книге компакт-диске содержится подробный пример, демонстрирующий эту технику.

МОДИФИКАЦИЯ ДИАЛОГОВОГО ОКНА USERFORM

На данном этапе предполагается, что диалоговое окно `UserForm` уже настроено. Вам осталось добавить элемент управления `MultiPage`. Первая страница элемента управления `MultiPage` будет содержать все первоначальные элементы управления. На второй странице располагаются элементы управления, которые используются для отображения индикатора текущего состояния. Когда макрос начнет выполняться, в коде VBA значение свойства `Value` элемента управления `MultiPage` изменится. В результате будут скрыты исходные элементы управления и отображены элементы управления, которые используются для создания индикатора текущего состояния.

Первым шагом будет добавление элемента управления `MultiPage` в диалоговое окно `UserForm`. После этого необходимо переместить все существующие в диалоговом окне `UserForm` элементы управления на первую страницу элемента управления `MultiPage`.

Затем следует активизировать вторую страницу элемента управления `MultiPage` и настроить ее так, чтобы она выглядела, как показано на рис. 15.2. В данном случае используется та же комбинация элементов управления, что и в предыдущем примере.

1. Добавьте элемент управления `Frame` и назовите его `FrameProgress`.
2. Добавьте элемент управления `Label` внутрь элемента управления `Frame` и назовите его `LabelProgress`. Удалите заголовок элемента управления `Label` и в качестве фона установите красную заливку.
3. Добавьте еще один элемент управления `Label`, чтобы отобразить описание выполняющейся задачи (этот элемент управления добавлять не обязательно).

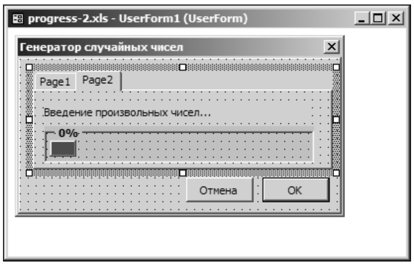


Рис. 15.2. Вторая вкладка элемента управления `MultiPage`, которая используется для отображения индикатора текущего состояния

4. После этого активизируйте элемент управления `MultiPage` (элемент управления целиком, а не одну из его вкладок) и установите его свойство `Style` в значение `2 - fmTabStyleNone` (это приведет к скрытию вкладок элемента управления). Самым простым способом выбора элемента управления `MultiPage` является использование раскрывающегося списка в диалоговом окне Properties. Скорее всего, понадобится изменить размеры объекта `MultiPage`, чтобы учесть отсутствие ярлыков вкладок.

ВСТАВКА ПРОЦЕДУРЫ UPDATEPROGRESS

Вставьте следующую процедуру в модуль кода диалогового окна UserForm.

```
Sub UpdateProgress(Pct)
    With UserForm1
      .FrameProgress.Caption = Format(Pct, "0%")
      .LabelProgress.Width = Pct * (.FrameProgress.Width - 10)
      .Repaint
    End With
End Sub
```

Эта процедура вызывается из основного макроса. Она выполняет фактическое обновление индикатора текущего состояния.

ИЗМЕНЕНИЕ ПРОЦЕДУРЫ

Далее необходимо модифицировать процедуру, которая выполняется при щелчке пользователя на кнопке OK. Данная процедура выступает обработчиком события Click и называется OKButton_Click. Для начала необходимо вставить оператор в начало процедуры.

```
MultiPage1.Value = 1
```

Этот оператор приводит к активизации второй вкладки элемента управления MultiPage (той страницы, на которой отображается индикатор текущего состояния).

На следующем шаге необходимо самостоятельно создать код, который будет подсчитывать степень завершения выполняемой задачи. Полученное значение следует присвоить переменной PctDone. Скорее всего, расчеты будут производиться внутри цикла. После этого добавьте приведенный ниже оператор, который будет обновлять индикатор текущего состояния.

```
Call UpdateProgress(PctDone)
```

КАК ЭТО РАБОТАЕТ

Данная методика довольно проста, кроме того, она требует использования только одного диалогового окна UserForm. В коде программа переходит к другой странице элемента управления MultiPage и превращает нормальное диалоговое окно в индикатор текущего состояния.

Отображение индикатора текущего состояния без использования элемента управления MultiPage

Пример, приведенный в этом разделе, подобен методу, представленному в предыдущем разделе. Но рассмотренная далее методика немного проще, так как не требует применения элемента управления MultiPage. Вместо этого индикатор текущего состояния содержится в нижней части диалогового окна UserForm. Для того чтобы элементы управления, составляющие индикатор текущего состояния, изначально не были видны, высота диалогового окна UserForm была уменьшена до соответствующего размера. Как только приходит время отображать индикатор текущего состояния, высота диалогового окна UserForm должна увеличиться, что сделает индикатор видимым на экране.

НА КОМПАКТ-ДИСКЕ

На прилагаемом к книге компакт-диске содержится пример, демонстрирующий использование этой методики.

На рис. 15.3 показано диалоговое окно `UserForm` в редакторе VBE. Свойство `Height` диалогового окна `UserForm` имеет значение 170. Но перед тем, как отобразить диалоговое окно `UserForm`, значение свойства `Height` устанавливается равным 120 (это приводит к тому, что элементы управления, представляющие индикатор текущего состояния, становятся невидимыми). Как только пользователь щелкнет на кнопке OK, код VBA изменит значение свойства `Height` на 170. Для этого используется следующий оператор.

```
Me.Height = 170
```

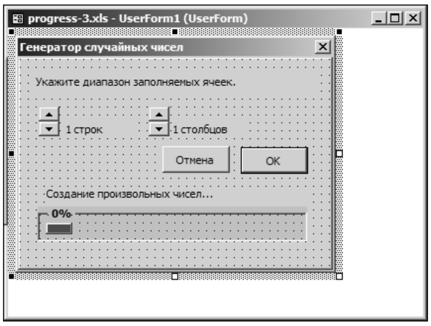


Рис. 15.3. Уменьшение высоты диалогового окна `UserForm` приведет к скрытию элементов управления, составляющих индикатор текущего состояния

Создание мастеров

Некоторые приложения используют специальные мастера для предоставления пользователям пошаговых инструкций по выполнению определенных задач. Мастер импорта текстовых файлов Excel является хорошим примером такого подхода к решению задач. Этот мастер является последовательностью диалоговых окон, которые предоставляют пользователю информацию и запрашивают у него необходимые сведения. Часто выбор пользователя в первых диалоговых окнах влияет на содержимое последующих окон. Как правило, пользователю предоставляется возможность свободно перемещаться вперед и назад по последовательности диалоговых окон. Кроме того, он может щелкнуть на кнопке Готово, чтобы использовать значения, принятые по умолчанию.

Вы вправе также создать "мастер" посредством VBA-кода и использования последовательности диалоговых окон `UserForm`. Однако существует более эффективный способ создания мастера с помощью единственного диалогового окна `UserForm` и элемента управления `MultiPage`.

На рис. 15.4 показан пример простого мастера из четырех этапов. Этот мастер состоит из диалогового окна `UserForm`, которое содержит элемент управления `MultiPage`. Каждый этап работы мастера соответствует отдельной вкладке элемента управления `MultiPage`.

Если вам необходимо создать мастер, то пример рабочей книги на прилагаемом компакт-диске может послужить хорошей отправной точкой. Это полноценный мастер из четырех этапов, который получает информацию от пользователя и размещает ее на листе.

В следующем разделе вы ознакомитесь с примером создания полноценного приложения-"мастера".

Настройка элемента управления MultiPage

Начните с создания нового диалогового окна `UserForm`. После этого добавьте элемент управления `MultiPage`. По умолчанию он содержит две страницы. Щелкните правой кнопкой мыши на элементе управления `MultiPage` и вставьте достаточное количество страниц, которые будут использоваться при создании мастера (по одной странице на каждый этап работы мастера). Пример мастера на компакт-диске состоит из четырех этапов, поэтому элемент управления `MultiPage` содержит четыре страницы. Имена страниц элемента управления `MultiPage` в данном случае роли не играют. Свойство `Style` элемента управления `MultiPage` должно быть установлено в значение **`2 - fmTabStyleNone`**. При настройке диалогового окна `UserForm` вкладки (страницы) необходимо сделать видимыми (чтобы иметь возможность обращаться к разным страницам элемента управления `MultiPage`).

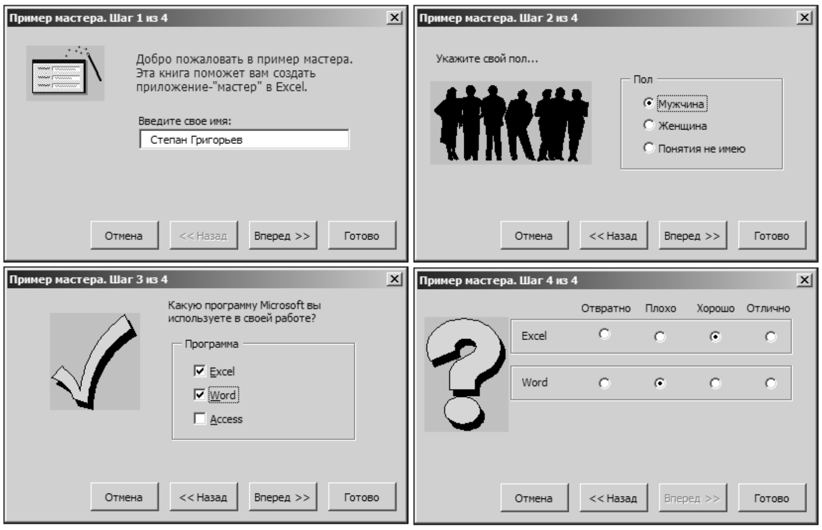


Рис. 15.4. Этот мастер, состоящий из четырех этапов, создан с использованием элемента управления `MultiPage`

Добавьте все необходимые элементы управления на каждую страницу элемента управления `MultiPage`. Эти элементы будут зависеть от целей конкретного приложения. Вы также вправе изменить размер элемента управления `MultiPage`, чтобы обеспечить место для всех элементов управления.

Добавление кнопок

Далее необходимо добавить кнопки, которые будут управлять переходом между этапами работы мастера. Эти кнопки должны размещаться за пределами элемента управления `MultiPage`, поскольку они используются при отображении любой страницы элемента управления `MultiPage`. Как правило, мастера имеют четыре стандартные кнопки.

- Отмена: позволяет отменить работу мастера.
- Назад: позволяет перейти к предыдущему этапу работы мастера. На первом этапе данная кнопка должна быть неактивна.

♦ Вперед: позволяет перейти к следующему этапу работы мастера. На последнем этапе эта кнопка должна быть неактивна.

♦ Готово: позволяет завершить работу мастера.

В некоторых случаях пользователю позволяется щелкнуть на кнопке Готово в любой момент работы мастера, что приводит к использованию значений, принятых по умолчанию. В других случаях мастер требует ответа пользователя на некоторые вопросы. Если возникла именно такая ситуация, то кнопка Готово должна быть отключена до того, как будут введены все необходимые данные. Пример на компакт-диске требует ввода информации в текстовое поле на первом этапе работы мастера.

В этом примере элементы управления `CommandButton` (кнопки) в коде называются `CancelButton`, `BackButton`, `NextButton` и `FinishButton`.

Программирование кнопок

Каждая из четырех кнопок мастера требует использования процедуры обработки события `Click`. Ниже приведена процедура обработки события для кнопки `CancelButton`. Эта процедура использует функцию `MsgBox` (рис. 15.5), чтобы проверить, завершена ли работа мастера. Если пользователь щелкнет на кнопке Отмена, то диалоговое окно `UserForm` будет выгружено из памяти, и никакие действия произведены не будут. Этот тип проверки не является обязательным.

```
Private Sub CancelButton_Click()
    Msg = "Прекратить работу мастера?"
    Ans = MsgBox(Msg, vbQuestion + vbYesNo, APPNAME)
    If Ans = vbYes Then Unload Me
End Sub
```

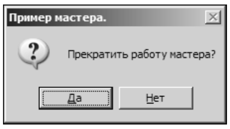


Рис. 15.5. Щелчок на кнопке Отмена *приводит к отображению окна сообщения*

Ниже представлены процедуры обработки событий для кнопок Далее и Назад.

```
Private Sub BackButton_Click()
    MultiPage1.Value = MultiPage1.Value - 1
    UpdateControls
End Sub

Private Sub NextButton_Click()
    MultiPage1.Value = MultiPage1.Value + 1
    UpdateControls
End Sub
```

Эти две процедуры очень просты. Они изменяют значение свойства `Value` элемента управления `MultiPage`, после чего вызывают другую процедуру, которая называется `UpdateControls` (данная процедура будет показана ниже).

Процедура `UpdateControls`, представленная в листинге 15.1, отвечает за включение и отключение кнопок `BackButton` и `NextButton`.

Листинг 15.1. Процедуры включения кнопок, управляющих работой мастера

```
Sub UpdateControls()
    Select Case MultiPage1.Value
        Case 0
            BackButton.Enabled = False
            NextButton.Enabled = True
        Case MultiPage1.Pages.Count - 1
            BackButton.Enabled = True
            NextButton.Enabled = False
        Case Else
            BackButton.Enabled = True
            NextButton.Enabled = True
    End Select

'   Обновить заголовок
    Me.Caption = APPNAME & " Шаг " _
      & MultiPage1.Value + 1 & " из " _
      & MultiPage1.Pages.Count

'   Поле Имя заполнять обязательно
    If tbName.Text = "" Then
        FinishButton.Enabled = False
    Else
        FinishButton.Enabled = True
    End If
End Sub
```

Процедура изменяет заголовок диалогового окна `UserForm`, в результате он отображает текущий этап работы мастера и общее количество этапов (константа `APPNAME` является глобальной и определена в модуле кода `Module1`). После этого проверяется содержимое поля Имя на первой странице элемента управления `MultiPage` (для создания этого поля используется элемент управления `TextBox`, который называется `tbName`). Данное поле следует обязательно заполнить, поэтому кнопка Готово отключена до момента заполнения. Если элемент управления `TextBox` остается пустым, кнопка `FinishButton` будет оставаться отключенной. В противном случае кнопка Готово активизируется, и у пользователя появляется возможность на ней щелкнуть.

Программирование зависимостей

В большинстве мастеров ответ пользователя на определенном этапе может повлиять на элементы управления, которые отображаются на последующих этапах. В примере на прилагаемом к книге компакт-диске на третьем этапе пользователь должен указать, какие программы он применяет в своей работе. После этого (на четвертом этапе) пользователю предлагается оценить выбранные программные продукты Microsoft. Элемент управления `OptionButton` для каждого продукта отображается только в том случае, если пользователь выбрал этот продукт на предыдущем этапе.

С точки зрения программирования, эта задача реализуется в результате обработки события `Change` элемента управления `MultiPage`. Как только значение элемента управления `MultiPage` изменится (после щелчка пользователя на кнопке Назад или Вперед), будет запущена процедура `MultiPage1_Change`. Если в элементе управления активной является последняя страница (четвертый этап), то процедура проверяет значения элементов управления `CheckBox` на странице, соответствующей третьему этапу работы мастера. После этого на странице для четвертого этапа выполняются необходимые изменения элементов управления.

В данном примере используется два массива — один для элементов управления `CheckBox`, соответствующих продуктам (используется на третьем этапе), а второй для элементов управления `Frame` (используется на четвертом этапе). Цикл `For Next`

скрывает элементы управления Frame для тех продуктов, которые не были выбраны на предыдущем этапе. После этого изменяется вертикальное расположение отображаемых на экране элементов управления Frame. Если на странице, соответствующей третьему этапу работы мастера, не был выбран ни один из продуктов, то на последнем этапе скрываются все элементы управления, кроме TextBox, который содержит сообщение Щелкните на кнопке Готово для выхода (если, конечно, на первом этапе введено имя). Процедура MultiPage1_Change приведена в листинге 15.2.

Листинг 15.2. Отображение страницы, соответствующей выбору пользователя

```
Private Sub MultiPage1_Change()
'   Настраивать страницу рейтинга?
    If MultiPage1.Value = 3 Then
'       Создание массива элементов управления CheckBox
        Dim ProdCB(1 To 3) As MSForms.CheckBox
        Set ProdCB(1) = cbExcel
        Set ProdCB(2) = cbWord
        Set ProdCB(3) = cbAccess

'       Создание массива элементов управления Frame
        Dim ProdFrame(1 To 3) As MSForms.Frame
        Set ProdFrame(1) = FrameExcel
        Set ProdFrame(2) = FrameWord
        Set ProdFrame(3) = FrameAccess

        TopPos = 22
        FSpace = 8
        AtLeastOne = False

'       Просмотр всех продуктов
        For i = 1 To 3
            If ProdCB(i) Then
                ProdFrame(i).Visible = True
                ProdFrame(i).Top = TopPos
                TopPos = TopPos + ProdFrame(i).Height + FSpace
                AtLeastOne = True
            Else
                ProdFrame(i).Visible = False
            End If
        Next i

'       Ни один из продуктов не выбран?
        If AtLeastOne Then
            lblHeadings.Visible = True
            Image4.Visible = True
            lblFinishMsg.Visible = False
        Else
            lblHeadings.Visible = False
            Image4.Visible = False
            lblFinishMsg.Visible = True
            If tbName = "" Then
                lblFinishMsg.Caption = _
                  "Введите имя в этапе 1."
            Else
                lblFinishMsg.Caption = _
                  "Щелкните на кнопке Готово для выхода."
            End If
        End If
    End If
End Sub
```

Выполнение задачи

Когда пользователь щелкает на кнопке Готово, мастер выполняет свою задачу: перемещает информацию из диалогового окна UserForm в следующую пустую строку рабочего листа. Эта процедура, показанная в листинге 15.3, довольно проста. Она начинается с определения следующей пустой строки рабочего листа и задания значения переменной (r). Остальная часть процедуры выполняет идентификацию значений элементов управления и ввод данных в ячейки листа.

Листинг 15.3. Вставка полученных данных в рабочий лист

```
Private Sub FinishButton_Click()
    r = Application.WorksheetFunction. _
      CountA(Range("A:A")) + 1

'   Вставить имя
    Cells(r, 1) = tbName.Text

'   Вставить пол
    Select Case True
        Case obMale: Cells(r, 2) = "Муж"
        Case obFemale: Cells(r, 2) = "Жен"
        Case obNoAnswer: Cells(r, 2) = "Другое"
    End Select


'   Определить используемость
    Cells(r, 3) = cbExcel
    Cells(r, 4) = cbWord
    Cells(r, 5) = cbAccess

'   Вставить оценки
    If obExcel1 Then Cells(r, 6) = ""
    If obExcel2 Then Cells(r, 6) = 0
    If obExcel3 Then Cells(r, 6) = 1
    If obExcel4 Then Cells(r, 6) = 2
    If obWord1 Then Cells(r, 7) = ""
    If obWord2 Then Cells(r, 7) = 0
    If obWord3 Then Cells(r, 7) = 1
    If obWord4 Then Cells(r, 7) = 2
    If obAccess1 Then Cells(r, 8) = ""
    If obAccess2 Then Cells(r, 8) = 0
    If obAccess3 Then Cells(r, 8) = 1
    If obAccess4 Then Cells(r, 8) = 2

'   Выгрузить форму
    Unload Me
End Sub
```

Как только мастер будет испытан и все станет работать должным образом, можно изменить значение свойства Style элемента управления MultiPage. Это свойство должно иметь значение **2 - fmTabStyleNone**.

Эмуляция функции MsgBox

Функция VBA MsgBox достаточно необычная — в отличие от остальных функций, она отображает диалоговое окно. С другой стороны, она, как и другие функции, также возвращает значение — целое число, представляющее кнопку, на которой щелкнул пользователь.

Приведенный ниже пример содержит пользовательскую функцию, которая эмулирует функцию VBA `MsgBox`. Вначале может показаться, что создание такой функции является простой задачей. Учитывая то, что функция `MsgBox` невероятно гибкая (благодаря огромному количеству аргументов, которые она принимает), можно утверждать: создать пользовательскую функцию, эмулирующую поведение функции `MsgBox`, непросто.

Основная наша задача заключается не в создании альтернативной функции для отображения окон сообщений — необходимо продемонстрировать разработку достаточно сложной функции, которая использует диалоговое окно `UserForm`. Кроме того, некоторым пользователям нравится сама идея создания функций для отображения окон сообщений. Созданную функцию легко модифицировать. Например, вы можете изменить используемый шрифт, цвет, надписи на кнопках и т.д.

Создаваемая функция, которая будет эмулировать поведение функции `MsgBox`, получила называние `MyMsgBox`. Эмуляция не является безупречной, так как функция `MyMsgBox` имеет следующие ограничения.

- Данная функция не поддерживает аргумент `Справка` (он способствует появлению кнопки Справка, щелчок на которой позволяет открыть файл справочной системы).
- Эта функция не поддерживает аргумент `Раздел` (он указывает раздел в файле справки).
- Данная функция не поддерживает опцию модальности (которая приостанавливает работу всей системы до тех пор, пока пользователь не ответит на запрос окна сообщения).

Функция `MyMsgBox` имеет следующий синтаксис.

```
MyMsgBox(Запрос[, Кнопки] [, Заголовок])
```

Этот синтаксис полностью соответствует синтаксису функции `MsgBox`, кроме того, что в первой не поддерживаются два последних аргумента (`Справка` и `Раздел`). Функция `MyMsgBox` использует те же предопределенные константы, что и функция `MsgBox`: `vbOKOnly`, `vbQuestion`, `vbDefaultButton1` и т.д.

НА ЗАМЕТКУ

С листингом функции `MsgBox` можно ознакомиться в диалоговом справочном руководстве, где представлена более подробная информация о ее аргументах.

Код функции MyMsgBox

Функция `MyMsgBox` использует диалоговое окно `UserForm`, которое называется `MyMsgBoxForm`. Данная функция очень короткая. Ее текст приводится ниже. Основная часть работы выполняется в функции обработки события `Initialize` (`UserForm_Initialize`).

НА КОМПАКТ-ДИСКЕ

Полный текст кода функции `MyMsgBox` слишком велик для размещения его на страницах книги, поэтому он представлен в файле рабочей книги на прилагаемом компакт-диске.

```
Public Prompt1 As String
Public Buttons1 As Integer
Public Title1 As String
Public UserClick As Integer
```

```
Function MyMsgBox(ByVal Prompt As String, _
  Optional ByVal Buttons As Integer, _
  Optional ByVal Title As String) As Integer
    Prompt1 = Prompt
    Buttons1 = Buttons
    Title1 = Title
    MyMsgBoxForm.Show
    MyMsgBox = UserClick
End Function
```

На рис. 15.6 показано действие функции MyMsgBox (для текста сообщения использован другой шрифт).

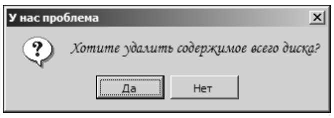


Рис. 15.6. Результат эмуляции функции MsgBox (в данном случае используется другой шрифт)

Ниже приведен код, который запускает функцию.

```
Prompt = "Хотите удалить содержимое всего диска?"
Buttons = vbQuestion + vbYesNo
Title = "У нас проблема"
Ans = MyMsgBox(Prompt, Buttons, Title)
```

Как это работает

Обратите внимание на использование переменных с глобальной областью действия. Первые три переменные (Prompt1, Buttons1 и Title1) представляют аргументы, которые передаются функции. Еще одна переменная (UserClick) указывает значения, возвращаемые функцией. Процедура UserForm_Initialize нуждается в способе получения этой информации и отправке ее обратно в функцию. Использование глобальных переменных (Public) является единственной возможностью реализации необходимого механизма.

Диалоговое окно UserForm (рис. 15.7) содержит четыре элемента управления Image (по одному на каждую возможную пиктограмму), три элемента управления CommandButton, а также элемент управления TextBox.

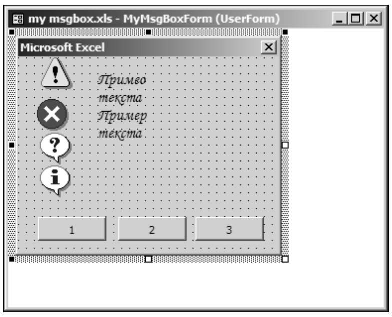


Рис. 15.7. Диалоговое окно UserForm, которое используется в функции MyMsgBox

Процедура `UserForm_Initialize` проверяет значения аргументов и выполняет следующие действия.

- Определяет, какое изображение необходимо вывести на экран (остальные изображения скрываются).
- Определяет, какие кнопки необходимо вывести на экран (остальные кнопки скрываются).
- Определяет, какая кнопка является выбранной по умолчанию.
- Выравнивает кнопки в диалоговом окне по центру.
- Определяет подписи для элементов управления `CommandButton`.
- Определяет расположение текста в диалоговом окне.
- Определяет необходимую ширину диалогового окна (используется функция API, которая предоставляет информацию о разрешении экрана).
- Определяет необходимую высоту диалогового окна.
- Отображает диалоговое окно `UserForm`.

Три дополнительные процедуры обработки событий используются для элементов управления `CommandButton`. Эти процедуры определяют, на какой из кнопок щелкнул пользователь. После этого присваивается соответствующее значение переменной `UserClick`.

Интерпретация второго аргумента (`Кнопки`) может оказаться немного сложнее. Этот аргумент выглядит следующим образом.

```
vbYesNoCancel + VbQuestion + VbDefaultButton3
```

Данный аргумент создает окно сообщения с тремя кнопками (Да, Нет и Отмена), отображает пиктограмму со знаком вопроса и делает третью кнопку выбранной по умолчанию. Аргумент равен 547 (3+32+512). Основной трудностью в данном случае может оказаться получение трех фрагментов информации на основе одного числа. Решить данную проблему несложно: преобразуйте числа в двоичную форму и проверьте состояние отдельных битов этого аргумента. Например, число 547 в двоичной форме записывается как 1000100011. Двоичные цифры с 4 по 6 определяют отображаемую пиктограмму, цифры с 8 по 10 определяют отображаемые кнопки, а цифры 1 и 2 определяют кнопку, которая будет выбрана по умолчанию.

Использование функции MyMsgBox

Для того чтобы применить эту функцию в собственном проекте, экспортируйте модуль `MyMsgBoxMod` и диалоговое окно `MyMsgBoxForm`. Затем эти два файла можно импортировать в собственный проект.

Немодальное диалоговое окно

Большинство диалоговых окон, о которых речь шла выше, модальные, т.е. их необходимо удалять с экрана, прежде чем приступать к работе с окном приложения, находящимся под этим окном. Некоторые диалоговые окна являются немодальными. Это означает, что пользователь может продолжать работу в приложении, даже когда диалоговое окно отображено на экране.

Excel 2000 — это первая версия Excel, в которой поддерживаются пользовательские немодальные диалоговые окна. Таким образом, данная функциональность отсутствует в предыдущих версиях Excel.

Чтобы отобразить немодальное диалоговое окно `UserForm`, используйте следующий оператор.

```
UserForm1.Show vbModeless
```

Слово `vbModeless` является встроенной константой, которая имеет значение 0. Таким образом, представленный далее оператор будет идентичен предыдущему.

```
UserForm1.Show 0
```

На рис. 15.8 показано немодальное диалоговое окно, которое отображает информацию об активной ячейке. Если диалоговое окно представлено на экране, пользователь может продолжать работу с ячейками и перемещаться на другие листы.

Этот пример доступен на прилагаемом к книге компакт-диске.

Важным моментом в использовании немодального диалогового окна является определение времени, когда необходимо обновить содержимое диалогового окна. С этой целью в нашем примере используется два события рабочей книги: `SheetSelectionChange` и `SheetActivate`. Процедуры обработки этих событий находятся в модуле кода объекта `ThisWorkbook`.

Обратитесь к главе 18, чтобы получить дополнительную информацию о событиях.

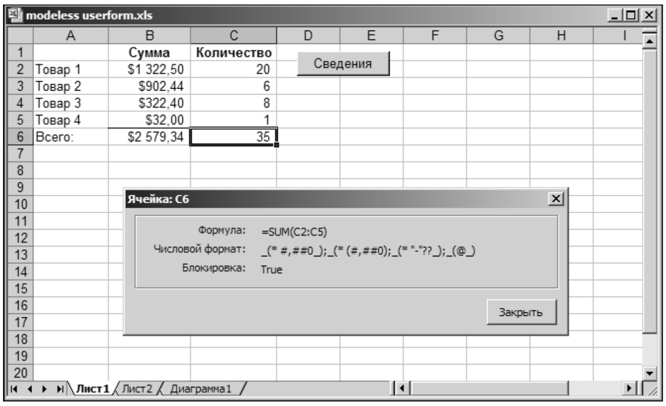


Рис. 15.8. Это немодальное диалоговое окно остается видимым на экране даже тогда, когда пользователь продолжает работать на рабочем листе

Ниже приведен код процедур обработки событий.

```
Private Sub Workbook_SheetSelectionChange _
  (ByVal Sh As Object, ByVal Target As Range)
    Call UpdateBox
End Sub

Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    Call UpdateBox
End Sub
```

Эти процедуры вызывают процедуру `UpdateBox`, которая представлена ниже.

```
Sub UpdateBox()
    With UserForm1
'       Проверка активного рабочего листа
        If TypeName(ActiveSheet) <> "Worksheet" Then
            .lblFormula.Caption = "N/A"
            .lblNumFormat.Caption = "N/A"
            .lblLocked.Caption = "N/A"
            Exit Sub
        End If

        .Caption = "Ячейка: " & ActiveCell.Address(False, False)
'       Формула
        If ActiveCell.HasFormula Then
            .lblFormula.Caption = ActiveCell.Formula
        Else
            .lblFormula.Caption = "(нет)"
        End If
'       Числовой формат
        .lblNumFormat.Caption = ActiveCell.NumberFormat
'       Блокировка
        .lblLocked.Caption = ActiveCell.Locked
    End With
End Sub
```

Процедура `UpdateBox` изменяет заголовок диалогового окна `UserForm`, который отображает адрес активной ячейки. После этого обновляются три элемента управления `Label` (`lblFormula`, `lblNumFormat` и `lblLocked`).

Ниже приведена информация, которая поможет понять, как работает этот код.

- Диалоговое окно `UserForm` отображается в немодальном режиме, что позволяет получать доступ к листу в течение того времени, пока окно отображается на экране. В Excel 97 и более ранних версиях немодальные диалоговые окна `UserForm` не поддерживаются.
- Код в верхней части процедуры проверяет, является ли рабочий лист активным. Если это не рабочий лист, то элементы управления `Label` получают заголовок N/A.
- Активная ячейка отслеживается благодаря событию `Selection_Change` (которое обрабатывается в модуле `ThisWorkbook`).
- Информация отображается в элементе управления `Label` пользовательской формы.

На рис. 15.9 показана более сложная версия диалогового окна рассмотренного примера (эту версию также можно найти на прилагаемом к книге компакт-диске). Данная версия отображает достаточно большое количество дополнительной информации о выделенной ячейке. Пользователи, которые давно используют Excel, могут заметить, что это диалоговое окно напоминает диалоговое окно Info (оно было удалено

из Excel несколько лет назад). Код этого примера слишком велик для того, чтобы приводить его в книге, однако отдельные комментарии все же стоит представлять.

- Диалоговое окно UserForm содержит флажок (Автоматическое обновление), который указывает на необходимость автоматического обновления диалогового окна UserForm.
- Рабочая книга использует модуль класса с целью обнаружить два события для всех открытых рабочих книг: SheetSelectionChange и SheetActivate. В результате каждый раз, когда возникают эти события, автоматически выполняется код, который отображает информацию об активной ячейке. Это происходит, если установлен параметр автоматического обновления. Некоторые действия, например, изменение формата значения в ячейке, не приводят к возникновению таких событий. Именно поэтому диалоговое окно содержит кнопку Обновить.
- Счетчики зависящих ячеек и ячеек, от которых зависит текущая, отображают данные только для активного листа. Это ограничение свойств Precedents и Dependents.
- Информация отображается в диалоговом окне UserForm с помощью элемента управления Label. Так как длина отображаемой информации может изменяться, код VBA изменяет размер и расстояние между элементами управления Label, а также размер самого диалогового окна UserForm в соответствии с длиной отображаемой информации.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Дополнительная информация о модулях классов изложена в главе 29.

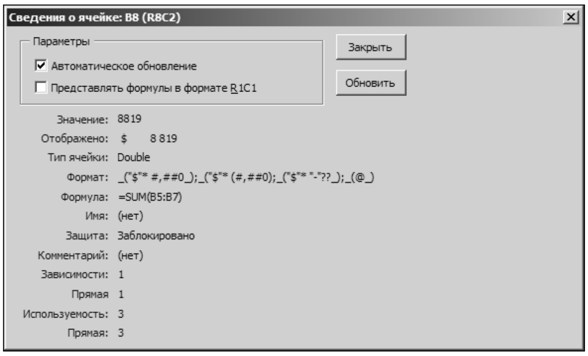


Рис. 15.9. Это диалоговое окно UserForm *отображает информацию об активной ячейке*

Несколько кнопок с одной процедурой обработки событий

Каждый элемент управления CommandButton в диалоговом окне UserForm должен иметь собственную процедуру обработки события Click. Например, если на форме находится два элемента управления CommandButton, то необходимо создать как минимум две процедуры обработки событий.

```
Private Sub CommandButton1_Click()
'Здесь располагается код
End Sub
```

```
Private Sub CommandButton2_Click()
'Здесь располагается код
End Sub
```

Другими словами, нельзя настроить макрос так, чтобы он выполнялся при щелчке на любой кнопке CommandButton. Каждая процедура обработки события Click жестко связана с определенным элементом управления CommandButton. Однако можно заставить каждую процедуру обработки события вызывать другой макрос — при этом следует передать параметр, который будет указывать, какая из кнопок нажата. В следующих примерах щелчок на одной из кнопок (CommandButton1 и CommandButton2) приведет к выполнению одной процедуры ButtonClick. Единственный аргумент сообщает процедуре ButtonClick, какая из кнопок была нажата.

```
Private Sub CommandButton1_Click()
     Call ButtonClick(1)
End Sub

Private Sub CommandButton2_Click()
     Call ButtonClick(2)
End Sub
```

Если в диалоговом окне UserForm находится несколько элементов управления CommandButton, то создание процедур обработки событий для каждого из элементов управления может оказаться утомительным. Целесообразно использовать единственную процедуру, которая определит, на какой из кнопок произведен щелчок, и в зависимости от этого будет выполнять соответствующие действия.

В данном разделе описывается способ, который поможет обойти это ограничение. Будет использован модуль класса с целью создания нового класса.

Этот пример доступен на прилагаемом к книге компакт-диске.

Процедура

Процесс создания примера рабочей книги состоит из следующих этапов.

1. Создайте пользовательское диалоговое окно и добавьте в него несколько элементов управления CommandButton (пример на компакт-диске содержит 16 элементов управления CommandButton). Предположим, что диалоговое окно называется UserForm1.
2. Вставьте в проект модуль класса (для этого необходимо выбрать команду Insert⇨Class Module), назовите его BtnClass и введите следующий код. Вам следует модифицировать процедуру ButtonGroup_Click.

```
Public WithEvents ButtonGroup As MsForms.CommandButton

Private Sub ButtonGroup_Click()
    Msg = "Вы щелкнули на " & ButtonGroup.Name & vbCrLf _
     & vbCrLf
    Msg = Msg & "Название: " & ButtonGroup.Caption _
     & vbCrLf
    Msg = Msg & "Расстояние слева: " & ButtonGroup.Left _
     & vbCrLf
    Msg = Msg & "Расстояние сверху: " & ButtonGroup.Top
    MsgBox Msg, vbInformation, ButtonGroup.Name
End Sub
```

3. Вставьте обычный модуль VBA и введите приведенный ниже код. Эта процедура используется для отображения диалогового окна `UserForm`.

   ```
   Sub ShowDialog()
       UserForm1.Show
   End Sub
   ```

4. В модуле кода диалогового окна `UserForm` введите код из листинга 15.4. Данная процедура будет запускаться при возникновении события `Initialize` для диалогового окна `UserForm`. Обратите внимание, что в коде исключается "реакция" на кнопку с названием `OKButton`. Таким образом, щелчок на кнопке `OKButton` не приведет к вызову процедуры `ButtonGroup_Click`.

Листинг 15.4. Создание массива объектов *Button*

```
Dim Buttons() As New BtnClass

Private Sub UserForm_Initialize()
    Dim ButtonCount As Integer
    Dim ctl As Control

'   Создание объектов Button
    ButtonCount = 0
    For Each ctl In UserForm1.Controls
        If TypeName(ctl) = "CommandButton" Then
            If ctl.Name <> "OKButton" Then 'Пропуск OKButton
                ButtonCount = ButtonCount + 1
                ReDim Preserve Buttons(1 To ButtonCount)
                Set Buttons(ButtonCount).ButtonGroup = ctl
            End If
        End If
    Next ctl
End Sub
```

После выполнения этих инструкций можно запустить процедуру `ShowDialog`, чтобы отобразить диалоговое окно `UserForm`. Щелчок на одной из кнопок `CommandButton` (кроме кнопки OK) приведет к выполнению процедуры `ButtonGroup_Click`. На рис. 15.10 показано окно сообщения, которое появляется после щелчка на одной из кнопок.

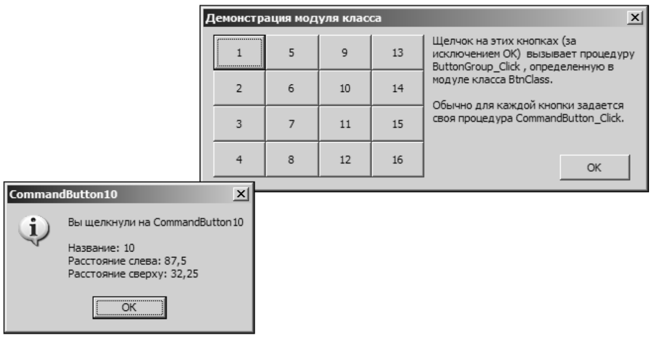


Рис. 15.10. Процедура `ButtonGroup_Click` выводит сообщение с информацией об используемой кнопке

Данную методику можно легко адаптировать для работы и с другими элементами управления. Необходимо изменить имя типа в объявлении `Public WithEvents`. Например, если вместо элементов управления `CommandButton` используются элементы управления `OptionButton`, то воспользуйтесь следующим оператором объявления объектов.

```
Public WithEvents ButtonGroup As MsForms.OptionButton
```

Диалоговое окно выбора цвета

Следующий пример напоминает рассмотренный в предыдущем разделе, но он немного сложнее. В данном случае рабочая книга демонстрирует использование диалогового окна `UserForm`, предоставляющего пользователю возможность выбора цвета на палитре рабочей книги (эта палитра состоит из 56 цветов).

Пример представлен одной функцией (`GetAColor`), которая отображает диалоговое окно `UserForm` и возвращает выбранное значение цвета.

НА КОМПАКТ-ДИСКЕ

Данный пример также содержится на прилагаемом к книге компакт-диске.

Ниже приведен код функции `GetAColor`.

```
Public ColorValue As Variant
Dim Buttons(1 To 56) As New ColorButtonClass

Function GetAColor() As Variant
'   Отображение формы UserForm и возвращение
'   значения цвета или False, если цвет не выбран
    Dim ctl As Control
    Dim ButtonCount As Integer
    ButtonCount = 0
    For Each ctl In UserForm1.Controls
'       Здесь указано 56 кнопок цветов'
'       Свойство Tag установлено в значение "ColorButton"
        If ctl.Tag = "ColorButton" Then
            ButtonCount = ButtonCount + 1
            Set Buttons(ButtonCount).ColorButton = ctl
'           Получение цветов из активной рабочей книги
            Buttons(ButtonCount).ColorButton.BackColor = _
                 ActiveWorkbook.Colors(ButtonCount)
        End If
    Next ctl
    UserForm1.Show
    GetAColor = ColorValue
End Function
```

Диалоговое окно `UserForm` содержит 56 элементов управления `CommandButton`, которые соответствуют различным цветам, полученным из палитры активной рабочей книги.

Доступ к функции `GetAColor` можно получить с помощью следующего выражения.

```
UserColor = GetAColor()
```

После выполнения этого выражения будет отображено диалоговое окно `UserForm`, а значение цвета назначено переменной `UserColor`. Это значение соответствует цвету, выбранному пользователем.

На рис. 15.11 показано диалоговое окно UserForm (в цвете оно выглядит лучше), которое содержит 56 элементов управления CommandButton. Свойство BackColor каждого элемента управления соответствует одному из цветов палитры активной рабочей книги. Щелчок на одной из кнопок приводит к выгрузке диалогового окна из памяти и передаче значения вызывающей функции.



Рис. 15.11. Это диалоговое окно позволяет пользователю выбрать цвет, щелкнув на кнопке

Файл примера, представленного прилагаемом к книге компакт-диске, содержит следующие компоненты.

- Пользовательское диалоговое окно (UserForm1), которое включает 56 элементов управления CommandButton, а также еще несколько элементов управления.
- Модуль класса (ColorButtonClass), который определяет класс ColorButton.
- Модуль VBA (Module1), который содержит процедуру GetAColor.
- Два примера, которые демонстрируют использование процедуры GetAColor.

Процедура GetAColor настраивает диалоговое окно UserForm и отображает его на экране. После этого процедура возвращает значение, соответствующее кнопке, на которой щелкнул пользователь. Если пользователь щелкает на кнопке Отмена, то процедура GetAColor возвращает значение False. Когда пользователь перемещает указатель мыши над кнопкой цвета, область Образец отображает цвет, соответствующий этой кнопке.

Код, который используется в этом диалоговом окне, является достаточно большим, поэтому в данном издании он не приводится. Однако вы можете открыть рабочую книгу, которая находится на прилагаемом компакт-диске, и непосредственно с ним ознакомиться.

Отображение диаграммы в пользовательском диалоговом окне

В Excel 5 или Excel 95 можно было легко отобразить “интерактивную” диаграмму в пользовательском диалоговом окне (для этого использовался диалоговый лист): достаточно было скопировать диаграмму и вставить ее на лист. Как ни удивительно, но в диалоговом окне UserForm не существует способа непосредственного отображения диаграммы. Можно, конечно, скопировать диаграмму и добавить ее в свойство Picture элемента управления Image, но это приведет к созданию статического изображения диаграммы, что не позволяет интерактивно отображать изменения, которые вносятся в исходные данные. Хотя диалоговые окна UserForm функционально превосходят старые диалоговые листы, разработчиками компании Microsoft диаграммам в диалоговых окнах не было уделено должного внимания.

> В Excel 97 и более поздних версиях можно продолжать использовать диалоговые листы. Таким образом, у пользователей существует возможность отображать интерактивные диаграммы в диалоговых окнах с помощью диалоговых листов.

Этот раздел содержит описание методов отображения диаграмм в диалоговых окнах `UserForm`.

Метод 1: сохранение диаграммы в виде файла

Тот факт, что Microsoft не позволяет отображать интерактивные диаграммы в диалоговом окне `UserForm`, совершенно не означает, что этого нельзя добиться! На рис. 15.12 показано диалоговое окно `UserForm`, в котором диаграмма отображена в виде объекта `Image`. На самом деле диаграмма находится на листе, а диалоговое окно `UserForm` отображает самое последнее состояние диаграммы. Описанная методика реализована благодаря копированию диаграммы во временный файл, после чего файл указывается в качестве свойства `Picture` элемента управления `Image`.



Рис. 15.12. Немного волшебства — и диалоговое окно `UserForm` отображает интерактивную диаграмму

ОСНОВНЫЕ ЭТАПЫ

Для того чтобы отобразить диаграмму в диалоговом окне `UserForm`, следуйте приведенным ниже инструкциям.

1. Создайте диаграмму или диаграммы обычным образом.
2. Вставьте диалоговое окно `UserForm` и добавьте в него элемент управления `Image`.
3. Создайте код VBA, который будет сохранять диаграмму в виде файла в формате GIF. После этого в коде необходимо установить свойство `Picture` элемента управления `Image` равным этому файлу. Вам следует воспользоваться функцией VBA `LoadPicture`.
4. Добавьте остальные элементы. Например, диалоговое окно `UserForm` в файле примера содержит элементы управления, которые позволяют изменять тип отображаемой диаграммы. С другой стороны, можно создать код для отображения нескольких диаграмм.

СОХРАНЕНИЕ ДИАГРАММЫ В ВИДЕ ФАЙЛА В ФОРМАТЕ GIF

Следующий код демонстрирует создание файла в формате GIF (который называется `temp.gif`) на основе диаграммы (в нашем случае первого объекта диаграммы на листе `Data`).

```
Set CurrentChart = Sheets("Data").ChartObjects(1).Chart
Fname = ThisWorkbook.Path & "\temp.gif"
CurrentChart.Export FileName:=Fname, FilterName:="GIF"
```

При выполнении этого кода будет показано всплывающее окно, которое отображает текущее состояние задачи. К сожалению, я не могу ответить на вопрос, как отключить отображение этого диалогового окна.

ИЗМЕНЕНИЕ СВОЙСТВА PICTURE ЭЛЕМЕНТА УПРАВЛЕНИЯ IMAGE

Если элемент управления `Image` в диалоговом окне `UserForm` называется `Image1`, то следующий оператор загрузит изображение (представленное переменной `Fname`) в элемент управления `Image`.

```
Image1.Picture = LoadPicture(Fname)
```

НА ЗАМЕТКУ

Данная методика работает корректно, но можно заметить небольшую задержку при сохранении и последующей загрузке диаграммы. В высокопроизводительных системах такую задержку заметить намного сложнее.

Метод 2: использование элемента управления OWC ChartSpace

Как отмечалось в главе 13, диалоговое окно `UserForm` может содержать другие элементы управления, которые обычно не представлены в диалоговом окне Toolbox. Компания Microsoft включила в пакет Office XP компоненты Office Web Components (OWC), которые можно использовать в собственных диалоговых окнах `UserForm`. На рис. 15.13 показан пример диалогового окна `UserForm`, в которое включен элемент управления `ChartSpace`.

НА ЗАМЕТКУ

Данная методика не позволяет отображать существующую диаграмму Excel в диалоговом окне `UserForm`. Вместо этого можно создать код, который будет рисовать диаграмму в элементе управления `ChartSpace`.

ПОЛУЧЕНИЕ ДОСТУПА К ЭЛЕМЕНТУ УПРАВЛЕНИЯ CHARTSPACE

Первым этапом является добавление элемента управления в диалоговое окно Toolbox. Щелкните правой кнопкой мыши на панели инструментов, чтобы отобразить диалоговое окно Additional Controls. Прокрутите список и установите флажок опции Microsoft Office Chart 10.0 (если используется Excel 2000, то эта опция будет называться Microsoft Office Chart 9.0). Щелкните на кнопке OK, и новый элемент управления будет размещен в диалоговом окне Toolbox.

ДОБАВЛЕНИЕ ЭЛЕМЕНТА УПРАВЛЕНИЯ CHARTSPACE В ДИАЛОГОВОЕ ОКНО USERFORM

Добавление элемента управления `ChartSpace` в диалоговое окно `UserForm` осуществляется подобно тому, как вставляются стандартные элементы управления. После добавления элемента управления в диалоговое окно диаграмма все еще не будет отображаться. Добавлен всего лишь элемент области диаграммы. Для создания самой диаграммы необходимо написать соответствующий код.

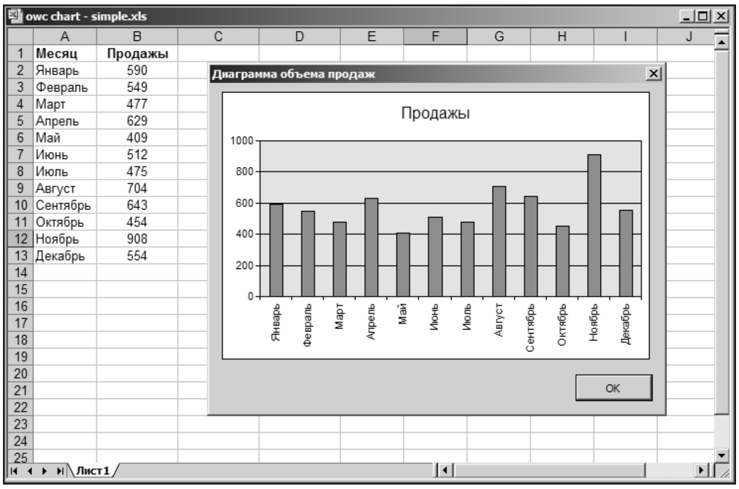


Рис. 15.13. Это диалоговое окно `UserForm` *содержит элемент управления* `ChartSpace`

СОЗДАНИЕ ДИАГРАММЫ

Следующий код, который находится в модуле диалогового окна `UserForm`, создает диаграмму на основе данных, которые хранятся в ячейках рабочего листа. Подписи категорий находятся в диапазоне `A2:A13`, а данные диаграммы содержатся в диапазоне `B2:B13`. Предполагается, что элемент управления `ChartSpace` называется `ChartSpace1`.

```
Sub CreateChart()
    Dim Chart1 As ChChart 'WCChart
    Dim Series1 As ChSeries 'WCSeries
    Dim r As Integer
    Dim XValues(1 To 12)
    Dim DataValues(1 To 12)

'   Добавить диаграмму в ChartSpace
    Set Chart1 = ChartSpace1.Charts.Add

'   Присвоить диаграмме имя
    With Chart1
        .HasTitle = True
        .Title.Caption = Range("B1")
    End With

    For r = 2 To 13
        XValues(r - 1) = Cells(r, 1)
        DataValues(r - 1) = Cells(r, 2)
    Next r

'   Создать ряды диаграммы
    Set Series1 = Chart1.SeriesCollection.Add

'   Указать тип диаграммы и данные
    With Series1
        .Type = chChartTypeColumnClustered
        .SetData chDimCategories, chDataLiteral, XValues
        .SetData chDimValues, chDataLiteral, DataValues
    End With
End Sub
```

Этот код начинается с объявления переменных. Если используется Excel 2000, то ст*о*ит обратить внимание на то, что типы объектов называются по-другому. Например, объект диаграммы имеет тип `WCChart` (а не `chChart`), подобно тому, как объект ряда диаграммы имеет тип `WCSeries` (а не `chSeries`). В коде используется два массива: один для хранения подписей категорий (`XValues`), а другой для хранения данных (`DataValues`).

Оператор `Set` создает объект `Chart`, который располагается в пределах элемента управления `ChartSpace`. Этот объект `Chart` называется `Chart1`. Следующий блок операторов устанавливает заголовок диаграммы, который извлекается из ячейки `B1`. Цикл `For Next` просматривает данные рабочего листа и заносит их в соответствующие массивы.

Следующий оператор `Set` добавляет к диаграмме ряды, и объект `Series` получает имя `Series1`. Блок кода `With-End With` указывает тип диаграммы (стандартная гистограмма) и данные для рядов.

Документацию по объектам OWC можно найти на локальном жестком диске компьютера. Файлы справки сохраняются на диске при установке компонентов OWC. Для получения дополнительной информации о свойствах и методах этих элементов управления можно воспользоваться окном Object Browser.

Важно понимать, что объектная модель для создания диаграммы с помощью OWC не соответствует объектной модели для создания диаграммы в Excel. В главе 18 описаны методы использования VBA для управления “настоящими” диаграммами Excel.

На рис. 15.14 показана более сложная версия данного примера. В этом случае пользователь может выбирать, какие данные будут отображаться на диаграмме. Кроме того, такая версия позволяет экспортировать диаграмму в виде файла формата GIF.

Это приложение доступно на прилагаемом к книге компакт-диске. Для его запуска вам потребуется Excel версии 2002 и выше.

Применение пакета Office Web Components

Office Web Components разрабатывались для создания интерактивных Web-страниц. В число этих компонентов входят Spreadsheet, Chart и Pivot Table. При создании приложения, которое использует OWC, необходимо установить пакет OWC на своем компьютере.

Пакет OWC поставляется в составе Microsoft Office 2000-2003. Но его установка не всегда выполняется автоматически. Другими словами, нельзя точно предположить, что пользователи Microsoft Office 2000 будут гарантированно иметь в системе пакет OWC (например, они могли сознательно отказаться от установки этого пакета). Для того чтобы внести еще больше недоразумений, отметим, что версия Small Business пакета Microsoft Office 2000 вообще не содержит пакета OWC.

Таким образом, включать элементы управления OWC в проекты Excel следует очень осторожно. Если приложение будет широко распространяться, то лучше избегать использования компонентов OWC.

Отображение листа в пользовательском диалоговом окне

Вам не понравилась отображенная в диалоговом окне `UserForm` диаграмма? А как насчет отображения в пользовательском диалоговом окне целого листа?

Рис. 15.15 представляет пример диалогового окна `UserForm`, который содержит элемент управления Microsoft Office Spreadsheet 10.0. Этот элемент управления может располагать полностью интерактивным листом, в котором есть формулы и форматирование. Использование элемента управления `Spreadsheet` имеет ряд преимуществ по сравнению с использованием стандартных листов Excel: элемент управления может иметь ширину до 18278 столбцов и высоту до 262144 строк. Это примерно в 300 раз больше, чем может содержать стандартный лист Excel.

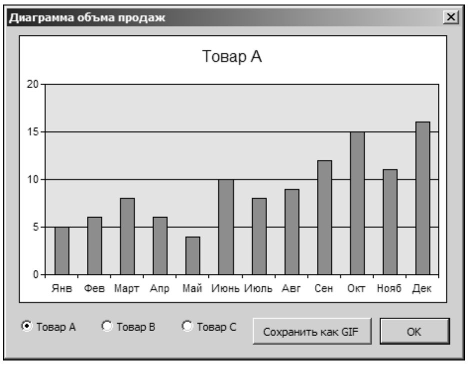


Рис. 15.14. Это диалоговое окно `UserForm` содержит элемент управления `ChartSpace` (который входит в состав пакета Office Web Components)

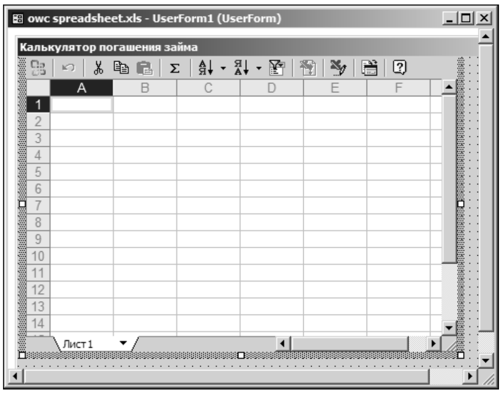


Рис. 15.15. Диалоговое окно `UserForm` содержит элемент управления `Spreadsheet`

ПОЛУЧЕНИЕ ДОСТУПА К ЭЛЕМЕНТУ УПРАВЛЕНИЯ SPREADSHEET

Для начала необходимо добавить элемент управления `Spreadsheet` в диалоговое окно Toolbox. Щелкните правой кнопкой мыши на панели инструментов, чтобы отобразить диалоговое окно Additional Control. Прокрутите список и установите флажок опции Microsoft Office Spreadsheet 11.0 (если используется Excel 2002, то этот элемент управления будет называться Microsoft Office Spreadsheet 10.0). Щелкните на кнопке OK, чтобы добавить этот элемент управления в диалоговое окно Toolbox.

ДОБАВЛЕНИЕ ЭЛЕМЕНТА УПРАВЛЕНИЯ SPREADSHEET В ДИАЛОГОВОЕ ОКНО USERFORM

Добавление элемента управления `Spreadsheet` в диалоговое окно `UserForm` осуществляется подобно тому, как добавляется любой другой элемент управления. Как только элемент управления будет добавлен в диалоговое окно `UserForm`, в этом окне появится таблица, состоящая из трех листов. Такую таблицу можно легко модифицировать.

ПРОСТОЙ ПРИМЕР

В приведенном далее примере использован элемент управления `Spreadsheet`, предназначенный для создания приложения расчета погашения займа. Завершенный проект показан на рис. 15.16. Пользователь может ввести информацию о займе в

столбце B, а ежемесячные выплаты рассчитываются (с помощью формулы) и отображаются в правой нижней ячейке.

Этот пример можно использовать только в демонстрационных целях. Применение элемента управления `Spreadsheet` сопряжено со слишком большими накладными расходами. Более эффективным подходом считается использование элементов управления `EditBox` и расчет выплат с помощью кода VBA.

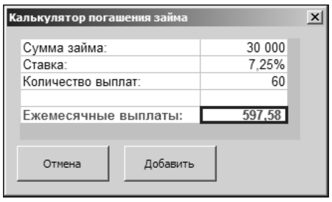


Рис. 15.16. Это диалоговое окно `UserForm` *использует элемент управления* `Spreadsheet` *для простого расчета выплат займа*

Добавление диалогового окна `UserForm` начните с создания новой рабочей книги, а далее следуйте приведенным ниже инструкциям. Удостоверьтесь, что к элементу управления `Spreadsheet` можно получить доступ из диалогового окна Toolbox.

1. Вставьте новое диалоговое окно `UserForm` и добавьте в него элемент управления `Spreadsheet`. Не меняйте имя этого элемента управления, используемое по умолчанию (`Spreadsheet1`).
2. По умолчанию таблица будет отображаться вместе с панелью инструментов, заголовками строк и столбцов, полосами прокрутки и вкладками листов. Чтобы освободить рабочее место, позже эти элементы управления будут отключены.
3. Выберите любую ячейку в элементе управления `Spreadsheet` и щелкните на ней правой кнопкой мыши. Из появившегося на экране контекстного меню выберите Команды и параметры.
4. Вы увидите диалоговое окно со вкладками (рис. 15.17).
5. Щелкните на вкладке Книга и удалите листы Лист2 и Лист3. После этого сбросьте флажки опций горизонтальную полосу прокрутки, вертикальную полосу прокрутки, ярлычки листов и панель инструментов.
6. В столбце A введите текст, показанный на рис. 15.16. После этого измените ширину столбца A, чтобы в нем свободно размещался только что введенный текст.
7. Введите параметры займа в диапазон `B1:B3`. После этого введите формулу в ячейку `B5`.

```
=ПЛТ(B2/12;B3;-B1)
```

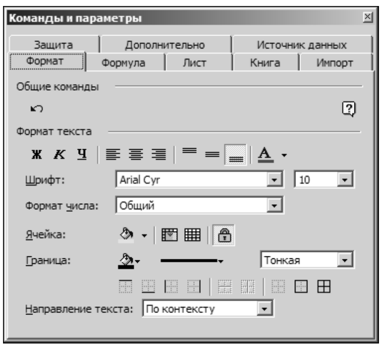


Рис. 15.17. Воспользуйтесь этим диалоговым окном для настройки элемента управления `Spreadsheet`

8. Выделите диапазон B1:B3 и щелкните на вкладке Формат в диалоговом окне Команды и параметры. Щелкните на пиктограмме в виде замка, что позволит разблокировать выделенные ячейки (остальные ячейки останутся заблокированными, что принято по умолчанию).
9. Щелкните на вкладке Лист в диалоговом окне Команды и параметры. В поле Видимый диапазон введите A1:B5.
10. Это приведет к тому, что будут скрыты все неиспользуемые ячейки за пределами указанного диапазона.
11. После этого сбросьте флажки заголовки строк и заголовки столбцов.
12. Наконец, добавьте два элемента управления CommandButton. Один (называющийся CancelButton) будет использоваться в качестве кнопки Отмена. Другой (PasteButton) будет выполнять код, который вставляет рассчитанные результаты в активные ячейки листа Excel.

Теперь пришло время добавить код VBA. На предыдущих этапах были разблокированы три ячейки. Блокирование ячеек не имеет никакого эффекта, пока лист не защищен (так же, как и в Excel). Таким образом, необходимо добавить код, который будет защищать лист при инициализации диалогового окна UserForm. Можно защитить лист на этапе проектирования (для этого используется диалоговое окно Команды и параметры), но тогда становится невозможным редактирование содержимого листа (если в лист вносятся изменения, то можно легко забыть включить его защиту). Защита листа на этапе выполнения позволяет удостовериться, что лист будет защищен и пользователь сможет вносить изменения только в ячейки, предназначенные для ввода.

Следующий код выполняет описанную выше задачу.

```
Private Sub UserForm_Initialize()
    Spreadsheet1.ActiveSheet.Protect
End Sub
```

Щелчок на кнопке PasteButton в диалоговом окне UserForm приведет к выполнению следующего кода.

```
Private Sub PasteButton_Click()
    ActiveCell.Value = Spreadsheet1.Range("B5")
    Unload Me
End Sub
```

Данная процедура размещает содержимое ячейки B5 (из элемента управления Spreadsheet) в активную ячейку рабочего листа Excel. После этого диалоговое окно UserForm выгружается из памяти.

Наконец, необходимо создать процедуру обработки события для кнопки Отмена.

```
Private Sub CancelButton_Click()
     Unload Me
End Sub
```

Не существует какой-либо особой причины для использования элемента управления Spreadsheet в диалоговом окне UserForm. Однако приятно осознавать, что если такая возможность появится, существует способ ею воспользоваться.

Расширенное диалоговое окно формы данных

Ниже приведен пример самого сложного диалогового окна `UserForm` среди всех тех, которые демонстрировались прежде. Настоящее диалоговое окно было разработано в качестве замены диалогового окна Excel Форма данных (рис. 15.18). Оно отображается на экране при выборе команды Данные⇨Форма.

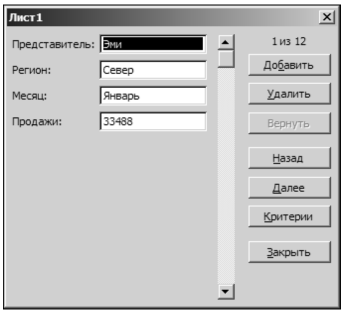


Рис. 15.18. Диалоговое окно Excel Форма данных

Как и диалоговое окно Форма данных, это диалоговое окно работает со списком на листе. Однако новое диалоговое окно выглядит совсем по-другому и предоставляет ряд дополнительных возможностей.

Описание

Новое диалоговое окно Форма данных содержит следующие дополнительные возможности.

- Обрабатывает любое количество записей и полей. Диалоговое окно Excel Форма данных поддерживает только 32 записи.
- Всегда имеет одинаковый размер, при этом обеспечивается прокрутка полей. Диалоговое окно Excel Форма данных не поддерживает прокрутку и может занять весь экран.
- Записи, которые отображены в диалоговом окне, всегда видны на экране и выделены другим цветом, что дает представление о текущем положении. Диалоговое окно Excel Форма данных не поддерживает прокрутку экрана и не выделяет текущую запись.
- После инициализации диалоговое окно всегда начинает работу с текущей активной ячейки. Диалоговое окно Excel Форма данных всегда начинает работу с первой записи в базе данных.
- При закрытии диалогового окна выделяется текущая запись. Диалоговое окно Excel Форма данных выделения записи не производит.
- Разрешается вставка записи в любую область базы данных. Диалоговое окно Excel Форма данных добавляет новые записи только в конец базы данных.
- Критерий поиска содержится на отдельной вкладке, что позволяет легко узнать о параметрах поиска. Диалоговое окно Excel Форма данных не всегда явно предоставляет информацию о критериях поиска.
- При поиске, в отличие от диалогового окна Excel Форма данных, поддерживается приблизительное совпадение (*, ? и #).

- Доступен полный исходный код VBA, что позволяет модифицировать диалоговое окно в соответствии с собственными требованиями. Диалоговое окно Excel Форма данных не создавалось с помощью VBA и не может быть модифицировано.



Расширенное диалоговое окно Форма данных является коммерческим продуктом (в некотором смысле), который может распространятся и использоваться бесплатно, но доступ к исходному коду предоставляется за определенную плату.

Установка надстройки

Для того чтобы воспользоваться расширенным диалоговым окном Форма данных, необходимо установить надстройку.

1. Скопируйте файл `dataform.xla` с компакт-диска на жесткий диск.
2. В Excel выберите команду Сервис⇨Надстройки.
3. В диалоговом окне Надстройки щелкните на кнопке Обзор и найдите файл `dataform.xla` в той папке, в которую он скопирован на первом шаге.

Использование расширенного диалогового окна формы данных

После установки надстройки станет доступна новая команда меню: Данные⇨JWalk Enhanced Data Form. Можно использовать соответствующее диалоговое окно для работы с любой базой данных, расположенной на листе.

Часть V

Совершенные методы программирования

В этой части...

Глава 16

Разработка утилит Excel с помощью VBA

В ЭТОЙ ГЛАВЕ...

В этой главе рассматриваются утилиты Excel. Основное назначение утилиты — расширение возможностей программы, обеспечение и облегчение доступа к уже представленным функциям программы.

- Об утилитах в общем и утилитах Excel в частности
- Цель использования VBA для разработки утилит
- Что необходимо знать при разработке утилиты
- Инструкции по разработке утилиты Excel для управления текстом в ячейках
- Поиск утилит для Excel

Вы увидите, что создание утилит Excel является отличным способом улучшения даже хорошего продукта.

Об утилитах Excel

Утилита не является конечным продуктом (как, например, приложение квартального отчета). Это инструмент, который позволяет создавать конечный продукт (например, квартальный отчет). Утилита Excel (почти всегда) — это надстройка, которая расширяет возможности Excel, добавляя в последнюю новые функциональные средства.

Excel, как и другие приложения Office, получает дополнительные функциональные возможности с выходом каждой новой версии. Однако многие пользователи со временем ощущают недостаток средств в существующем программном обеспечении. Например, пользователи, которым не нравится сетка на экране, требуют добавления в программу опции по ее скрытию. Ее можно было бы разместить в диалоговом окне настройки параметров, которое отображается на экране при выборе команды Сервис⇨Параметры. Можно также назначить одной из панелей инструментов соответствующую кнопку. Пользователи, часто работающие с датами, требуют добавления команды отображения всплывающего календаря, который упрощает введение дат в текущую ячейку. Отдельные пользователи нуждаются в более простом способе экспорта диапазона данных в файл.

С другой стороны, утилиты не должны быть слишком сложными. Многие из популярных утилит очень просты. Например, представленная ниже процедура VBA является утилитой, которая отображает и скрывает сетку в активном окне.

```
Sub ToggleGridDisplay()
     ActiveWindow.DisplayGridlines = _
         Not ActiveWindow.DisplayGridlines
End Sub
```

Этот макрос можно сохранить в персональной книге макросов, чтобы всегда иметь к нему доступ. Вы получите быстрый доступ к макросу, если назначите ему значок на панели инструментов и определите команду в меню или опцию в контекстном меню. Также макрос можно вызвать определенной комбинации клавиш.

Некоторые примеры, приведенные в части IV, на самом деле являются утилитами (или они легко могут быть преобразованы в утилиты).

Использование VBA для разработки утилит

При изучении бета-версии Excel 5 многих поразил потенциал VBA, который на световые годы опережал встроенный язык Excel для создания макросов XLM и делал Excel лидером среди программ управления электронными таблицами, особенно в вопросах программирования новых возможностей.

В процессе изучения VBA я создал коллекцию утилит Excel, которые были написаны исключительно с использованием VBA. Я понял, что могу изучить язык программирования быстрее, если поставлю перед собой реальную цель. Результатом стало создание пакета Power Utility Pak для Excel.

При изучении языка VBA следует учитывать такие моменты.

- VBA может оказаться трудным в изучении, но со временем его использование упрощается.
- Экспериментирование является основным способом изучения VBA. Проект обычно вырастает из десятка экспериментов по созданию кода, что в итоге приводит к появлению завершенного продукта.
- VBA предоставляет возможность использовать Excel таким образом, который соответствует внешнему виду и поведению Excel, включая меню, панели инструментов и диалогового окна.
- Excel может выполнять любые задачи. Если разработка зашла в тупик, то существует вероятность, что к решению ведет другой путь.

Немногие другие продукты содержат такой богатый набор инструментов, который позволяет конечному пользователю настолько значительно расширить возможности программы.

Из чего состоит хорошая утилита

Утилита Excel должна упрощать выполнение задачи, причем делать это более эффективно, чем смог бы произвести вручную пользователь. Если утилита разрабатывается для конечных пользователей, то что же делает ее ценной? Ниже приводится список составляющих хорошей утилиты.

- *Утилита добавляет что-либо к Excel.* Это может быть и новая возможность, и комбинация существующих функций, и более простой способ использования одного из существующих средств.
- *Утилита имеет универсальную природу.* В идеальном случае она должна обладать возможностями, которые работают в различных средах. Конечно, создать утилиту общего назначения намного сложнее, чем создать утилиту, работающую в строго определенной среде.

- *Утилита обладает гибкостью.* Лучшие утилиты предоставляют большое количество параметров, что позволяет использовать их для обработки широкого диапазона ситуаций.
- *Утилита выглядит, работает и кажется встроенной командой Excel.* Если появится желание добавить собственные штрихи к внешнему виду создаваемых утилит, у других пользователей не возникнет трудностей с их использованием (однако утилиты должны выглядеть и вести себя так же, как и встроенные команды Excel).
- *Утилита должна предоставлять пользователям необходимую справочную информацию.* Другими словами, следует создать документацию, к которой пользователь может обратиться в случае необходимости.
- *Утилита отслеживает появление ошибок.* Конечный пользователь не должен увидеть сообщение об ошибке VBA. Любое сообщение об ошибке, которое увидит пользователь, должно создаваться автором утилиты.
- *Утилита должна предоставлять способ отмены результата собственных действий.* Пользователи, которых не удовлетворяет результат работы утилиты, должны иметь возможность отменить внесенные изменения.

Текстовые инструменты: анатомия утилит

В этом разделе рассматривается утилита Excel, которая была разработана в качестве части пакета Power Utility Pak. Утилита Text Tools позволяет пользователю манипулировать текстом в выделенном диапазоне ячеек. В частности, эта утилита поддерживает такие операции.

- Изменение регистра текста (верхний регистр, нижний регистр или правильный регистр).
- Добавление символов в начале, в конце или в указанной позиции.
- Удаление текста в начале, в конце или в указанной позиции строки.
- Удаление лишних пробелов (или всех пробелов)

На рис. 16.1 показана утилита Text Tools в действии.

Эту утилиту вы найдете на прилагаемом к книге компакт-диске. Она представляет собой отдельный инструмент, который также входит в состав пакета Power Utility Pak. Для установки данной надстройки воспользуйтесь стандартной методикой — выполните команду Сервис⇨Надстройки. Вы также можете открыть ее файл с помощью команды Файл⇨Открыть.

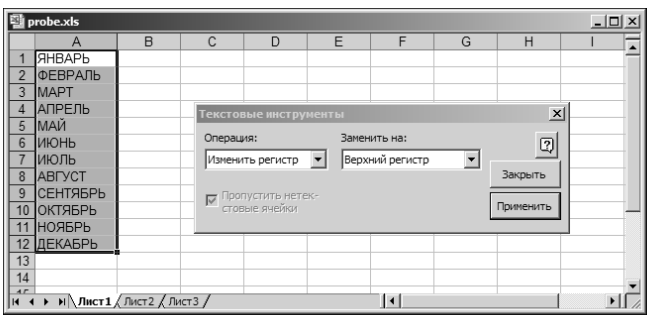


Рис. 16.1. Используйте эту утилиту для изменения регистра символов

Обоснование

Excel содержит ряд текстовых инструментов, позволяющих манипулировать текстовыми строками несколькими способами. Например, можно преобразовать символы в ячейке в верхний регистр или удалить из текста определенный символ. Также существует возможность удалить из строки пробелы и т.д. Но для того чтобы выполнить одну из этих операций, необходимо создать формулы, скопировать их, преобразовать в значения и вставить эти значения вместо первоначального текста. Другими словами, Excel не предоставляет простого способа управления текстом. Было бы неплохо, чтобы Excel имела инструменты работы с текстом, которые не требуют использования формул.

Кстати, многие прогрессивные идеи начинаются с фразы “Если бы...”.

Цели проекта утилиты Text Tools

Первым этапом в разработке утилиты является точное определение того, что будет делать утилита. Ниже приведен первоначальный план, представленный в виде десяти целей.

- Утилита должна иметь тот же внешний вид и поведение, что и остальные команды Excel. Другими словами, утилита будет отображать диалоговое окно, которое выглядит так же, как и другие диалоговые окна Excel.
- Доступ к утилите можно получить из меню Сервис.
- Утилита должна работать с выделенным диапазоном ячеек (включая множественное выделение) и предоставлять пользователю возможность модифицировать диапазон ячеек до тех пор, пока отображено диалоговое окно.
- Основные возможности утилиты заключаются в изменении регистра символов, добавлении нового текста в строки, удалении фиксированного количества символов из текста и удалении пробелов из текста в каждой ячейке.
- Кроме того, пользователю предоставляется возможность просмотра базовой статистики по выделенным ячейкам.
- Пользователь имеет возможность проводить перечисленные изменения как по отношению к текстовым, так и по отношению к нетекстовым ячейкам.
- Утилита не будет вносить изменения в ячейки, которые содержат формулы.
- Утилита должна быть быстрой и эффективной. Например, если пользователь выделит диапазон, утилита должна игнорировать пустые ячейки, которые присутствуют в этом диапазоне.
- Пользователю должна предоставляться возможность отмены внесенных изменений.
- Пользователь может получать справочные сведения из диалогового руководства.

Как работает утилита

Когда открывается рабочая книга утилиты Text Tools, создается новая команда в меню Сервис. Она называется Текстовые инструменты. Выбор этой команды приведет к запуску процедуры `RunTextTools`, которая проверяет состояние рабочей среды Excel (необходимо, чтобы рабочая книга была активна и не была защищена), после чего отображается диалоговое окно Текстовые инструменты.

Пользователь может выбрать необходимые опции для изменения текста и щелкнуть на кнопке Применить, чтобы применить их. Изменения сразу же будут представлены в рабочей книге. Диалоговое окно продолжает отображаться на экране. Каждую операцию можно отменить. Пользователь также имеет возможность выполнить неко-

торые дополнительные действия. Щелчок на кнопке Справка приведет к отображению диалогового окна справочного руководства. Щелчок на кнопке Закрыть вызывает закрытие диалогового окна.

Рабочая книга утилиты Text Tools

Рабочая книга утилиты Text Tools состоит из следующих компонентов.

- *Один лист.* Каждая рабочая книга должна иметь как минимум один лист. В данном случае этот лист используется для хранения процедур обработки операций отмены действий.
- *Один модуль кода VBA.* Этот модуль содержит объявления общедоступных переменных и констант. Кроме того здесь находится код отображения пользовательской формы и управления процедурой отмены действия. Весь код, выполняющий конечные действия, сохранен в модуле пользовательской формы.
- *Одно диалоговое окно UserForm.* Содержит пользовательскую форму.
- *Модуль кода рабочей книги.* Содержит код создания и удаления команды меню Сервис.

Пользовательская форма утилиты

Создание утилиты обычно начинается с разработки пользовательского интерфейса, который в данном случае определяется основным диалоговым окном. Операция по созданию диалогового окна заставляет повторно пересмотреть все концепции проекта.

Диалоговое окно `UserForm` содержит различное количество элементов управления, в зависимости от уже выбранных в окне параметров. На рис. 16.2 показана пользовательская форма в окне редактора VBE.

Вы можете заметить, что отдельные элементы управления располагаются не в тех местах, что при запуске и выполнении утилиты. Это происходит потому, что их расположение динамически изменяется в коде. Ниже описаны главные элементы диалогового окна.

- *Список Операция.* Этот список находится в левой части формы и предназначен для указания типа выполняемой операции.
- *Список Proc1.* Точно указывает текстовую операцию, соответствующую указанному в предыдущем списке типу.
- *Список Proc2.* Только две текстовые операции нуждаются в использовании этого списка. С его помощью указываются дополнительные инструкции для текстовой команды.

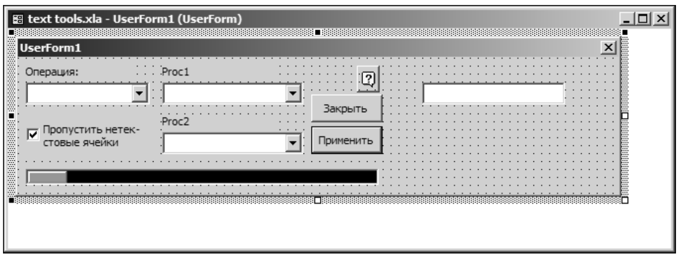


Рис. 16.2. Диалоговое окно `UserForm` содержит самые разные элементы управления

- *Флажок*. Позволяет игнорировать нетекстовые ячейки.
- *Кнопка Справка*. Это элемент управления `CommandButton`, который содержит изображение. Щелчок на указанной кнопке приводит к отображению справочной системы утилиты.
- *Кнопка Закрыть*. Щелчок на данной кнопке приведет к выгрузке пользовательской формы.
- *Кнопка* Применить. Щелчок на этом элементе управления `CommandButton` приводит к применению текстовых параметров, которые установлены в пользовательской форме.
- *Индикатор выполнения команды*. Состоит из элементов управления `Label` и `Frame`.
- *Текстовое поле*. Используется при добавлении текста.

На рис. 16.3 показана пользовательская форма при выборе самых разных текстовых инструментов. Обратите внимание на изменение параметров в окне при выборе другой команды.



Рис. 16.3. Содержание окна изменяется в зависимости от выбранных параметров

Вы можете заметить, что утилита нарушает одно из основных правил разработки, которое приводилось ранее в этой главе. В отличие от встроенных диалоговых окон Excel, пользовательское окно не содержит кнопки OK или Отмена, а по щелчку на кнопке Применить нельзя закрыть диалоговое окно. Первоначальная версия утилиты Text Tools содержала кнопку OK, и щелчок на ней приводил к выполнению необходимого действия и закрытию диалогового окна. Частое общение с пользователями доказало целесообразность изменения существующего интерфейса. Большинство людей предпочитают выполнять одновременно несколько операций над текстом в выделенных ячейках. Таким образом, утилита была изменена, чтобы больше соответствовать запросам конечных пользователей.

Модуль ThisWorkbook

При открытии надстройки запускается процедура `Workbook_Open`. Она добавляет в меню Сервис новую команду. Название этой команды меню определяется константой `APPNAME`, объявленной в модуле `Module1`.

```
Private Sub Workbook_Open()
    Dim ToolsMenu As CommandBarPopup
    Dim NewMenuItem As CommandBarButton
    Set ToolsMenu = Application.CommandBars(1).FindControl(Type:=10, ID:=30007)
    Set NewMenuItem = ToolsMenu.Controls.Add(Type:=1, Temporary:=True)
    With NewMenuItem
        .Caption = APPNAME
        .OnAction = "RunTextTools"
    End With
End Sub
```

При закрытии надстройки выполняется процедура `Workbook_BeforeClose`. С ее помощью из меню Сервис удаляется новая команда

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    On Error Resume Next
    Application.CommandBars(1).FindControl(Type:=10, _
               ID:=30007).Controls(APPNAME).Delete
    On Error GoTo 0
End Sub
```

На рис. 16.4 показано меню Сервис с вновь добавленной командой.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Детально операция добавления новых элементов меню рассмотрена в главе 23.

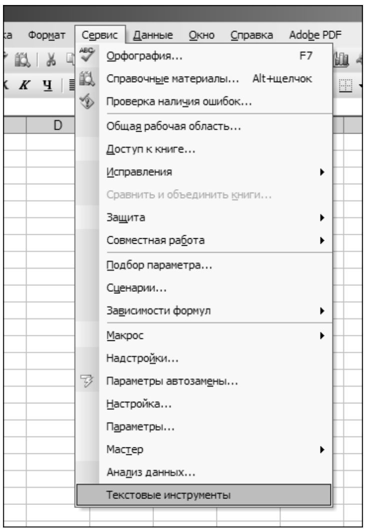


Рис. 16.4. В меню Сервис *добавлена новая команда*

Модуль Module1

Модуль `Module1` объявляет данные, содержит простую процедуру, которая завершает утилиту и управляет двумя операциями отмены.

ОБЪЯВЛЕНИЕ ДАННЫХ

Ниже приведены объявления, которые находятся в начале модуля кода `Module1`.

```
Public Const APPNAME As String = "Текстовые инструменты"
Public Const PROGRESSTHRESHHOLD = 2000
Public UserChoices(1 To 8) As Variant
Public UndoRange As Range
Public UserSelection As Range
```

Вначале объявляется глобальная константа, которая содержит строку, хранящую имя приложения. Эта строка используется в окнах сообщений и в качестве заголовка создаваемой команды меню.

Константа `PROGRESSTHRESHHOLD` указывает количество ячеек, для которых отображается индикатор выполнения операции. В нашем случае индикатор отображается только при обработке более 2000 ячеек.

Массив `UserChoices` содержит значения каждого элемента управления. Эта информация сохраняется в системном реестре и запрашивается при выполнении утилиты.

Кроме того, здесь представлены два объекта `Range`. Их назначение — хранить информацию, необходимую для отмены выполненной операции.

ПРОЦЕДУРА RUNTEXTTOOLS

Ниже приведена процедура `RunTextTools`.

```
Sub RunTextTools()
    Dim InvalidContext As Boolean
    If ActiveSheet Is Nothing Then InvalidContext = True
    If TypeName(ActiveSheet) <> "Worksheet" Then InvalidContext = True
    If InvalidContext Then
        MsgBox "Выберите ячейки в диапазоне.", vbCritical, APPNAME
    Else
        UserForm1.Show 0
    End If
End Sub
```

Очевидно, что эта процедура простая. Она проверяет, активен ли лист, а затем еще раз проверят его принадлежность к рабочим листам. Если хотя бы одно условие не справедливо, то переменной `InvalidContext` назначается значение `True`. Конструкция `If-Then-Else` проверяет значение этой переменной и отображает сообщение или пользовательскую форму. Обратите внимание, что метод `Show` использует аргумент `0`, что указывает на немодальность формы.

НА ЗАМЕТКУ

В коде этой процедуры не проверяется, выделен ли на рабочем листе диапазон. Этот тип проверки включен в процедуру обработки события для кнопки Применить.

ПРОЦЕДУРА UNDOTEXTTOOLS

Эта процедура выполняется при выборе в Excel команды Правка⇨Отмена (или щелчке на панели инструментов на кнопке Отмена). Она описана дальше в этой главе.

Модуль UserForm1

Все важные операции выполняются с помощью кода, сохраненного в модуле UserForm1. Ниже вкратце описаны все процедуры этого модуля. Сам код модуля слишком длинный, чтобы приводить его, но вы всегда можете открыть файл text tools.xla на прилагаемом к книге компакт-диске и ознакомиться с ним.

ПРОЦЕДУРА USERFORM_INITIALIZE

Выполняется перед отображением пользовательской формы. Она изменяет размер формы, чтобы поместить в ней все необходимые параметры (их значения сохраняются в системном реестре). Данная процедура также определяет список элементов, которые указывают выполняемую операцию.

ПРОЦЕДУРА COMBOBOXOPERATION

Выполняется после выбора пользователем операции в раскрывающемся списке. Скрывает и отображает элементы управления.

ПРОЦЕДУРА APPLYBUTTON_CLICK

Выполняется при щелчке на кнопке Применить. Вначале она выполняет проверку данных, а затем вызывает функцию CreateWorkRange. В результате в рабочий диапазон не включаются пустые ячейки. Процедура ApplyButton_Click также вызывает процедуру SaveForUndo, которая сохраняет текущее состояние на случай отмены выполненных операций.

Далее в процедуре используется конструкция Select Case, с помощью которой определяется процедура, выполняющая указанную пользователем операцию: ChangeCase, AddText, RemoveText, RemoveSpaces или RemoveCharacters. Отдельные процедуры вызывают функции. Например, процедура ChangeCase вызывает процедуру ToggleCase или SentenceCase.

ПРОЦЕДУРА CLOSEBUTTON_CLICK

Выполняется при щелчке на кнопке Закрыть. Она сохраняет параметры текущего состояния в системном реестре и выгружает пользовательскую форму.

ПРОЦЕДУРА HELPBUTTON_CLICK

Выполняется при щелчке на кнопке Справка. Отображает содержимое справочного файла.

Повышение эффективности утилиты Text Tools

Процедуры утилиты выполняют возложенные на них задачи, циклически просматривая диапазон ячеек. При этом нет никакого смысла просматривать неизменяющиеся ячейки, например, пустые ячейки или ячейки с формулами.

Как уже отмечалось ранее, процедура ApplyButton_Click вызывает процедуру функции CreateWorkRange. Эта функция создает и возвращает объект Range, в котором нет пустых ячеек и ячеек с формулами. Например, пусть столбец A содержит текст в диапазоне A1:A12. Если пользователь выделит весь столбец, то функция CreateWorkRange преобразует диапазон всего столбца в диапазон без пустых ячеей (т.е. A1:A12). В результате приложение будет выполняться эффективнее, не проводя обработку ненужных (в данном случае пустых) данных.

Функция `CreateWorkRange` принимает два аргумента.

- `Rng`. Объект `Range`, представляющий выделенный пользователем диапазон данных.
- `TextOnly`. Булево значение. При равенстве его `True` функция возвращает ячейки только с текстовыми значениями. В противном случае возвращаются все непустые ячейки.

Ниже приведен код функции `CreateWorkRange`.

```
Private Function CreateWorkRange(Rng, TextOnly)
'   Создает и возвращает объект Range
    Set CreateWorkRange = Nothing

'   Одна ячейка, с формулой
    If Rng.Count = 1 And Rng.HasFormula Then
        Set CreateWorkRange = Nothing
        Exit Function
    End If

'   Одна ячейка, одна объединенная ячейка
    If Rng.Count = 1 Or Rng.MergeCells = True Then
        If TextOnly Then
            If Not IsNumeric(Rng(1).Value) Then
                Set CreateWorkRange = Rng
                Exit Function
            Else
                Set CreateWorkRange = Nothing
                Exit Function
            End If
        Else
            If Not IsEmpty(Rng(1)) Then
                Set CreateWorkRange = Rng
                Exit Function
            End If
        End If
    End If
    On Error Resume Next
    Set Rng = Intersect(Rng, Rng.Parent.UsedRange)
    If TextOnly = True Then
        Set CreateWorkRange = Rng.SpecialCells(xlConstants, xlTextValues)
        If Err <> 0 Then
            Set CreateWorkRange = Nothing
            On Error GoTo 0
            Exit Function
        End If
    Else
        Set CreateWorkRange = Rng.SpecialCells(xlConstants, xlTextValues + _
                           xlNumbers)
        If Err <> 0 Then
            Set CreateWorkRange = Nothing
            On Error GoTo 0
            Exit Function
        End If
    End If
End Function
```

НА ЗАМЕТКУ

Функция `CreateWorkRange` активно использует свойство `SpecialCell`. Для того чтобы получить о нем дополнительную информацию, попытайтесь в диалоговом окне Excel Выделение группы ячеек записать макрос при создании различных выделений. Это диалоговое окно можно отобразить, нажав клавишу <F5> и щелкнув на кнопке Выделить в диалоговом окне Перейти.

Диалоговое окно Выделение группы ячеек имеет одну особенность. Обычно диалоговое окно работает с текущим выделенным диапазоном. Например, если выделен целый столбец, то результатом будет подмножество ячеек этого столбца. Но если выделена одна ячейка, то диалоговое окно работает со всем листом. Именно поэтому функция `CreateWorkRange` проверяет количество ячеек, которые составляют диапазон, переданный функции в качестве аргумента.

Сохранение настроек утилиты Text Tools

Утилита Text Tools характеризуется одним очень интересным свойством: она сохраняет последние используемые настройки. Это невероятно удобно, поскольку чаще всего требуется повторно выполнять одни и те же действия.

Как уже отмечалось, последние настройки приложения сохраняются в системном реестре. После щелчка на кнопке Отмена выполняется функция `SaveSettings`, которая отвечает за сохранение значений каждого элемента управления. При запуске утилиты выполняется функция `GetSettings`, которая запрашивает настройки из системного реестра.

В системном реестре настройки утилиты сохраняются в следующем разделе.

```
HKEY_CURRENT_USER\Software\VB and VBA Program Settings\Text Tools Utility\Settings
```

На рис. 16.5 показана эта ветвь системного реестра (редактор реестра — это программа `regedit.exe`).

Методика отмены выполненных действий

В отличие от средства отмены действий, которое поддерживается в Excel, операция отмены выполненных действий, реализуемая в утилите Text Tools, является одноуровневой. Другими словами, пользователь может отменить только последнюю выполненную операцию.

Утилита сохраняет исходные данные в рабочем листе. Если пользователь отменяет операцию, то данные копируются обратно в пользовательскую рабочую книгу.

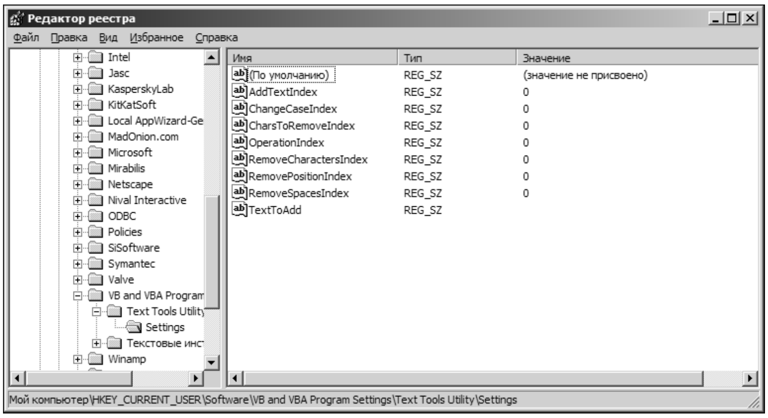


Рис. 16.5. С помощью этой программы можно легко просмотреть записи системного реестра

В утилите Text Tools в модуле `Module1` общедоступные данные объявляются следующим образом.

```
Public UndoRange As Range
Public UserSelection As Range
```

Перед измерением данных процедура `ApplyButtonClick` вызывает процедуру `SaveForUndo`. В начале процедуры определяются три переменные.

```
Set UserSelection = Selection
Set UndoRange = WorkRange
ThisWorkbook.Sheets(1).UsedRange.Clear
```

В данном случае `WorkRange` — это объект `Range`, который включает рабочий диапазон без пустых ячеек и ячеек с формулами. Третий оператор удаляет любые сохраненные данные.

После этого выполняется цикл.

```
For Each RngArea In WorkRange.Areas
    ThisWorkbook.Sheets(1).Range(RngArea.Address).Formula = RngArea.Formula
Next RngArea
```

В цикле просматривается содержимое объекта `WorkRange`, а его данные сохраняются в рабочем листе (если этот объект состоит только из одного непрерывного диапазона, то в нем анализируется всего одна область).

После успешного выполнения описанных выше операций в коде вызывается метод `OnUndo`. С его помощью определяется процедура отмены выполненных действий. Например, в случае изменения регистра текста этот оператор принимает следующий вид.

```
Applocation.OnUndo "Отмена изменения регистра", "UndoTextTools"
```

Меню Правка Excel дополнится новой командой. При ее выборе выполняется процедура `UndoTextTools`, приведенная ниже.

```
Private Sub UndoTextTools()
'   Отмена последней операции
    Dim a As Range
    Application.ScreenUpdating = False
    With UserSelection
        .Parent.Parent.Activate
        .Parent.Activate
        .Select
    End With
    For Each a In UndoRange.Areas
        a.Formula = ThisWorkbook.Sheets(1).Range(a.Address).Formula
    Next a
    Application.ScreenUpdating = True
    On Error GoTo 0
    Exit Sub
ErrHandler:
    Application.ScreenUpdating = True
    MsgBox "Нельзя отменить", vbInformation, APPNAME
    On Error GoTo 0
End Sub
```

НА КОМПАКТ-ДИСКЕ

На прилагаемом к книге компакт-диске содержится пример, который демонстрирует возможность использования команды Правка⇨Отмена после выполнения процедуры VBA.

Отмена действия процедуры VBA

Пользователи уже привыкли к возможности отмены выполненных операций. Почти каждая операция, которая выполняется в Excel, может быть отменена. А с появлением Excel 97 в программе поддерживается несколько уровней отмены действий.

В процессе программирования на VBA у вас возникнет вопрос, можно ли отменить результат выполнения процедуры. Ответ: да. Если быть более точным, то следует заметить, что добиться такого поведения не всегда легко.

Отмена результата выполнения процедуры VBA не реализуется по умолчанию. Процедура должна где-то сохранить информацию о предыдущем состоянии системы, чтобы его можно быть восстановить при выборе пользователем команды Правка⇨Отмена. Практическая реализация такого подхода зависит от действий, которые выполняет процедура. В крайнем случае вам понадобится сохранить полностью весь лист. Если процедура изменяет диапазон ячеек, то необходимо программно сохранять содержимое этого диапазона.

Объект `Application` содержит метод `OnUndo`, который позволяет программисту указать текст, появляющийся в меню Правка⇨Отмена, а также процедуру, которая выполняется при выборе пользователем команды Правка⇨Отмена. Например, следующий оператор определяет команду Отменить выполнение макроса. Если пользователь выберет опцию Правка⇨Отменить выполнение макроса, то будет запущена процедура `UndoMyMacro`.

```
Application.OnUndo "Отменить выполнение макроса", "UndoMyMacro"
```

Оценка проекта

В предыдущих разделах было представлено описание всех компонентов утилиты Text Tools. Пришло время пересмотреть первоначальные цели проекта, чтобы оценить возможность их достижения. Ниже приведены первоначальные цели проекта с дополнительными комментариями.

- *Утилита должна иметь тот же внешний вид и поведение, что и остальные команды Excel. Другими словами, утилита будет предоставлять пользователю диалоговое окно, которое выглядит так же, как и другие диалоговые окна Excel.* Как было отмечено ранее, утилита Text Tools незначительно отходит от этого принципа, так как вместо кнопки OK в ней используется кнопка Применить. Принимая во внимание критерий повышения удобства использования, на такое нарушение принципов поведения окон Excel можно согласиться.
- *Доступ к утилите можно будет получить из меню* Сервис. Цель достигнута.
- *Утилита должна работать с текущим диапазоном ячеек (включая множественное выделение) и предоставлять пользователю возможность изменения диапазона ячеек, пока отображено диалоговое окно.* Цель достигнута.
- *Основные возможности утилиты будут состоять из инструментов изменения регистра символов, добавления нового текста в строки, удаления фиксированного количества символов из текста и удаления пробелов из текста в каждой ячейке.* Цель достигнута.
- *Пользователю должна предоставляться возможность запрашивать перечисленные типы изменений и по отношению к текстовым, и по отношению к нетекстовым ячейкам.* Цель достигнута.
- *Утилита не будет вносить изменения в ячейки, которые содержат формулы.* Цель достигнута.

- *Утилита будет быстрой и эффективной. Например, если пользователь выделяет диапазон, утилита игнорирует пустые ячейки, которые входят в этот диапазон.* Цель достигнута.
- *Пользователю будет предоставлена возможность отмены внесенных изменений.* Цель достигнута, но нестандартными методами.
- *Пользователю будет предоставлена возможность получения доступа к справочному руководству.* Цель достигнута.

Принципы работы утилиты Text Tools

Если вы не совсем разобрались с принципами работы этой утилиты, то загрузите рабочую книгу и воспользуйтесь отладчиком для пошагового просмотра кода. Целесообразно попытаться выполнить утилиту по отношению к различными выделенным диапазонами, включая весь рабочий лист. Вы заметите, что, независимо от размера диапазона, обрабатываются только текстовые ячейки, а пустые полностью игнорируются. Если на листе находится только одна текстовая ячейка, то утилита будет работать очень быстро независимо от того, выделена только эта ячейка или выделен весь рабочий лист.

Для получения наилучших результатов можно утилиту Text Tools лучше устанавливать как надстройку, а не открывать как файл. В главе 21 приведена информация о создании надстроек.

Глава 17

Работа со сводными таблицами

В ЭТОЙ ГЛАВЕ...

Сводные таблицы являются самым мощным средством Excel. Сводные таблицы впервые появились в Excel 5. Они поддерживаются только в Excel (пока ни один программный продукт для управления электронными таблицами не предоставляет подобную возможность). Эта глава не содержит вступительной информации по использованию сводных таблиц. Предполагается, что вы уже знакомы с данной темой (как и с методами ручного создания и изменения сводных таблиц).

- Что необходимо знать для создания сводной таблицы с помощью VBA.
- Примеры процедур VBA, которые используются для создания сводных таблиц.
- Пример использования VBA для изменения существующей сводной таблицы.

Как легко заметить, создание сводной таблицы на основе базы данных или списка позволяет быстро отобразить необходимые итоговые данные, которые другим способом представить на экране или бумаге просто невозможно. Кроме того, такое представление данных можно получить очень быстро. В Excel также поддерживается возможность создания кода VBA, который позволяет управлять сводными таблицами.

НА ЗАМЕТКУ

Поддержка сводных таблиц была значительно расширена в Excel 2000. В этой версии реализовано более эффективное кэширование данных, а также добавлена поддержка сводных диаграмм. Сводная диаграмма — это диаграмма, которая связана со сводной таблицей. Следовательно, часть материала в данной главе относится только к Excel 2000 и более поздним версиям.

Вступительный пример

Этот раздел начинается с описания простого примера использования кода VBA для создания сводной таблицы.

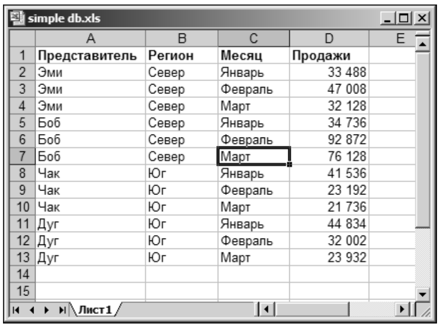
На рис. 17.1 показана простая база данных на рабочем листе. Она состоит из четырех полей: `Представитель`, `Регион`, `Месяц` и `Продажи`. Записи представляют объем продаж определенного торгового представителя за определенный месяц.

Создание сводной таблицы

На рис. 17.2 показана сводная таблица, которая создана на основе указанных данных. Эта сводная таблица обеспечивает отображение суммарного объема ежемесячных продаж каждого торгового представителя. Сводная таблица содержит следующие поля.

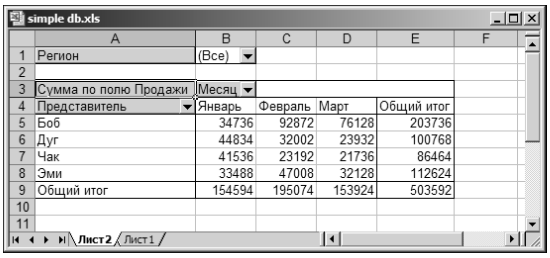
- `Регион` — поле страницы в сводной таблице.
- `Представитель` — поле строки в сводной таблице.

- Месяц — поле столбца в сводной таблице.
- Продажи — поле данных в сводной таблице, которое использует функцию Sum (СУММ).



Представитель	Регион	Месяц	Продажи
Эми	Север	Январь	33 488
Эми	Север	Февраль	47 008
Эми	Север	Март	32 128
Боб	Север	Январь	34 736
Боб	Север	Февраль	92 872
Боб	Север	Март	76 128
Чак	Юг	Январь	41 536
Чак	Юг	Февраль	23 192
Чак	Юг	Март	21 736
Дуг	Юг	Январь	44 834
Дуг	Юг	Февраль	32 002
Дуг	Юг	Март	23 932

Рис. 17.1. Эта простая база данных может служить основой сводной таблицы



Регион	(Все)			
Сумма по полю Продажи	Месяц			
Представитель	Январь	Февраль	Март	Общий итог
Боб	34736	92872	76128	203736
Дуг	44834	32002	23932	100768
Чак	41536	23192	21736	86464
Эми	33488	47008	32128	112624
Общий итог	154594	195074	153924	503592

Рис. 17.2. Сводная таблица, которая создана на основе данных, показанных на рис. 17.1

Перед созданием сводной таблицы была включена функция записи макроса. Далее представлен код, который был при этом сгенерирован.

```
Sub Macro1()
'   Записанный макрос
    Range("A1").Select
    ActiveWorkbook.PivotCaches.Add _
        (SourceType:=xlDatabase, _
        SourceData:="Лист1!R1C1:R13C4"). _
        CreatePivotTable _
        TableDestination:="", _
        TableName:="СводнаяТаблица1", _
        DefaultVersion:=xlPivotTableVersion10

    ActiveSheet.PivotTableWizard _
        TableDestination:=ActiveSheet.Cells(3, 1)
    ActiveSheet.Cells(3, 1).Select

    With ActiveSheet.PivotTables("СводнаяТаблица1"). _
        PivotFields("Регион")
        .Orientation = xlPageField
        .Position = 1
    End With

    With ActiveSheet.PivotTables("СводнаяТаблица1"). _
        PivotFields("Представитель")
        .Orientation = xlRowField
        .Position = 1
    End With

    With ActiveSheet.PivotTables("СводнаяТаблица1"). _
        PivotFields("Месяц")
        .Orientation = xlColumnField
        .Position = 1
    End With
    ActiveSheet.PivotTables("СводнаяТаблица1"). _
        AddDataField ActiveSheet.PivotTables("СводнаяТаблица1"). _
        PivotFields("Продажи"), "Сумма по полю Продажи", xlSum
End Sub
```

Сгенерированный при записи макроса код зависит от процесса создания сводной таблицы. В предыдущем примере мы создали сводную таблицу, которая оставалась пустой, пока поля не были перенесены с панели инструментов Сводные таблицы. Существует также альтернативный способ: щелкните на кнопке Макет на втором этапе работы мастера сводных таблиц и создайте макет сводной таблицы еще до ее создания.

Можно, конечно, выполнить записанный макрос для создания еще одной сводной таблицы, идентичной первой. Однако прежде убедитесь, что лист с данными активен в момент создания сводной таблицы.

Просмотр созданного кода

Код VBA, записанный при создании сводной таблицы, может привести вас в замешательство. Для того чтобы разобраться в записанном макросе, потребуется информация, содержащаяся в интерактивном справочном руководстве.

- `PivotCaches` — коллекция объектов `PivotCache` в объекте `Workbook`.
- `PivotTables` — коллекция объектов `PivotTable` в объекте `Worksheet`.
- `PivotFields` — коллекция полей в объекте `PivotTable`.
- `PivotItems` — коллекция отдельных элементов данных в поле.
- `CreatePivotTable` — метод объекта `PivotCache`, который создает сводную таблицу на основе данных, содержащихся в кэш-памяти.
- `PivotTableWizard` — метод объекта `Worksheet`, который создает сводную таблицу (использование этого метода не является необходимым — см. следующий раздел).

Очистка записанного кода

Как и в случае с большинством записанных макросов, предыдущий пример не настолько эффективен, как следовало ожидать. Его можно упростить, чтобы сделать немного понятнее. В листинге 17.1 содержится код, который генерирует сводную таблицу, подобную той, которая показана в предыдущем примере.

Листинг 17.1. Эффективный код VBA генерации сводной таблицы

```
Sub CreatePivotTable()
'   Вручную созданный макрос
    Dim PTCache As PivotCache
    Dim PT As PivotTable

    Set PTCache = ActiveWorkbook.PivotCaches.Add _
        (SourceType:=xlDatabase, _
         SourceData:=Range("A1").CurrentRegion.Address)

    Set PT = PTCache.CreatePivotTable _
        (TableDestination:="", _
         TableName:="СводнаяТаблица1")

    With PT
        .PivotFields("Регион").Orientation = xlPageField
        .PivotFields("Месяц").Orientation = xlColumnField
        .PivotFields("Представитель").Orientation = xlRowField
        .PivotFields("Продажи").Orientation = xlDataField
    End With
End Sub
```

Процедура `CreatePivotTable` упрощена (что делает ее более простой в понимании), так как в ней объявляются две переменные объектов: `PivotCache` и `PT`. Эти переменные заменяют собой ссылки `ActiveSheet.PivotCaches` и `ActiveSheet.PivotTables`. Новый объект `PivotCache` создается с помощью метода `Add`. После этого создается объект `PivotTable`, для чего используется метод `CreatePivotTable` коллекции `PivotCaches`. Последняя часть кода добавляет поля к сводной таблице и указывает их расположение (поле страницы, столбца, строки или данных).

Обратите внимание, что первоначальный макрос жестко связан с диапазоном данных, который использовался при создании объекта `PivotCache` (в данном случае это диапазон `"Лист1!R1C1:R13C4"`). В процедуре `CreatePivotTable` сводная таблица основывается на текущем диапазоне, окружающем ячейку `A1`. Это гарантирует, что макрос будет продолжать корректно выполняться даже тогда, когда в диапазон добавлены дополнительные данные.

Данный код можно сделать более универсальным за счет использования индексов вместо названий в коллекции `PivotFields`. При таком подходе, если пользователь вносит изменения в заголовки столбцов, код продолжает выполняться. Например, в более универсальном макросе используется оператор `PivotFields(1)` вместо оператора `PivotFields("Регион")`.

Чтобы лучше понять рассматриваемую тему, запишите собственный макрос и изучите ключевые объекты, методы и свойства. После этого необходимо обратиться к разделами справочного руководства, чтобы разобраться, как все это работает. Практически во всех случаях необходимо вносить изменения в "свежезаписанный" макрос. Как только вам станут понятны принципы управления сводными таблицами, можете приступать к созданию кода без предварительной записи макроса.

Создание сложной сводной таблицы

В этом разделе представлен код VBA, который предназначен для создания относительно сложной сводной таблицы.

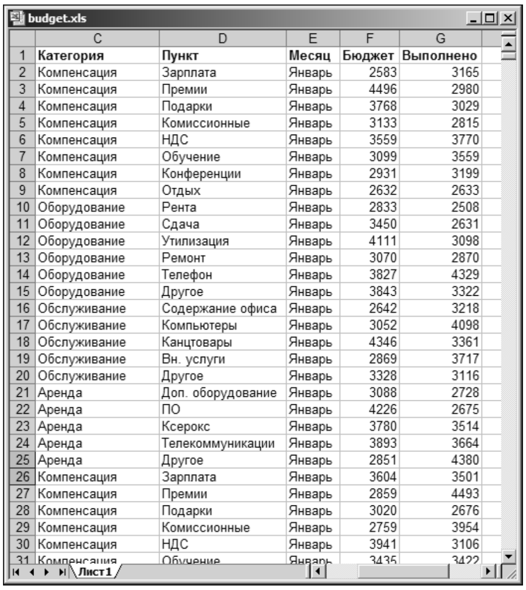
Данные

На рис. 17.3 показан фрагмент базы данных на листе. Эта таблица содержит 15840 строк иерархически упорядоченной информации о бюджете корпорации. В корпорации существует пять подразделений; каждое подразделение содержит по одиннадцать отделов. Отдел имеет четыре бюджетных категории, а каждая категория состоит из нескольких пунктов. Бюджетные и фактические расходы указываются для каждого (из двенадцати) месяца.

Эта рабочая книга доступна на прилагаемом к книге компакт-диске.

Сводная таблица

На рис. 17.4 показана сводная таблица, которая создана на основе приведенных выше данных. Обратите внимание на то, что сводная таблица содержит вычисляемое поле, которое называется `Разница`, а также четыре вычисляемых элемента (`Кв1`, `Кв2`, `Кв3` и `Кв4`), которые отображают квартальные суммы.

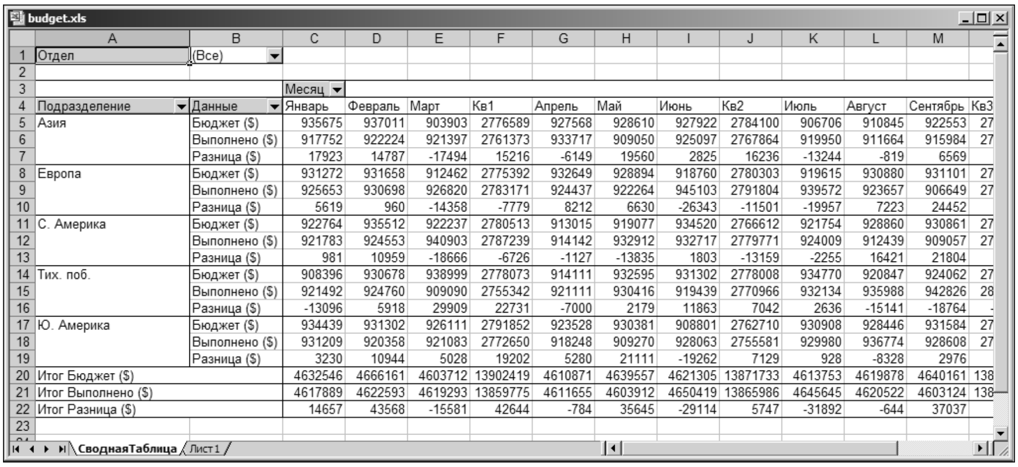


budget.xls

	C	D	E	F	G
1	**Категория**	**Пункт**	**Месяц**	**Бюджет**	**Выполнено**
2	Компенсация	Зарплата	Январь	2583	3165
3	Компенсация	Премии	Январь	4496	2980
4	Компенсация	Подарки	Январь	3768	3029
5	Компенсация	Комиссионные	Январь	3133	2815
6	Компенсация	НДС	Январь	3559	3770
7	Компенсация	Обучение	Январь	3099	3559
8	Компенсация	Конференции	Январь	2931	3199
9	Компенсация	Отдых	Январь	2632	2633
10	Оборудование	Рента	Январь	2833	2508
11	Оборудование	Сдача	Январь	3450	2631
12	Оборудование	Утилизация	Январь	4111	3098
13	Оборудование	Ремонт	Январь	3070	2870
14	Оборудование	Телефон	Январь	3827	4329
15	Оборудование	Другое	Январь	3843	3322
16	Обслуживание	Содержание офиса	Январь	2642	3218
17	Обслуживание	Компьютеры	Январь	3052	4098
18	Обслуживание	Канцтовары	Январь	4346	3361
19	Обслуживание	Вн. услуги	Январь	2869	3717
20	Обслуживание	Другое	Январь	3328	3116
21	Аренда	Доп. оборудование	Январь	3088	2728
22	Аренда	ПО	Январь	4226	2675
23	Аренда	Ксерокс	Январь	3780	3514
24	Аренда	Телекоммуникации	Январь	3893	3664
25	Аренда	Другое	Январь	2851	4380
26	Компенсация	Зарплата	Январь	3604	3501
27	Компенсация	Премии	Январь	2859	4493
28	Компенсация	Подарки	Январь	3020	2676
29	Компенсация	Комиссионные	Январь	2759	3954
30	Компенсация	НДС	Январь	3941	3106
31	Компенсация	Обучение	Январь	3435	3422

Лист1

Рис. 17.3. Данные в этой рабочей книге будут собраны в сводной таблице



budget.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M	
1	Отдел	(Все)												
2														
3			Месяц											
4	Подразделение	Данные	Январь	Февраль	Март	Кв1	Апрель	Май	Июнь	Кв2	Июль	Август	Сентябрь	Кв3
5	Азия	Бюджет ($)	935675	937011	903903	2776589	927568	928610	927922	2784100	906706	910845	922553	27
6		Выполнено ($)	917752	922224	921397	2761373	933717	909050	925097	2767864	919950	911664	915984	27
7		Разница ($)	17923	14787	-17494	15216	-6149	19560	2825	16236	-13244	-819	6569	
8	Европа	Бюджет ($)	931272	931658	912462	2775392	932649	928894	918760	2780303	919615	930880	931101	27
9		Выполнено ($)	925653	930698	926820	2783171	924437	922264	945103	2791804	939572	923657	906649	27
10		Разница ($)	5619	960	-14358	-7779	8212	6630	-26343	-11501	-19957	7223	24452	
11	С. Америка	Бюджет ($)	922764	935512	922237	2780513	913015	919077	934520	2766612	921754	928860	930861	27
12		Выполнено ($)	921783	924553	940903	2787239	914142	932912	932717	2779771	924009	912439	909057	27
13		Разница ($)	981	10959	-18666	-6726	-1127	-13835	1803	-13159	-2255	16421	21804	
14	Тих. поб.	Бюджет ($)	908396	930678	938999	2778073	914111	932595	931302	2778008	934770	920847	924062	27
15		Выполнено ($)	921492	924760	909090	2755342	921111	930416	919439	2770966	932134	935988	942826	28
16		Разница ($)	-13096	5918	29909	22731	-7000	2179	11863	7042	2636	-15141	-18764	-
17	Ю. Америка	Бюджет ($)	934439	931302	926111	2791852	923528	930381	908801	2762710	930908	928446	931584	27
18		Выполнено ($)	931209	920358	921083	2772650	918248	909270	928063	2755581	929980	936774	928608	27
19		Разница ($)	3230	10944	5028	19202	5280	21111	-19262	7129	928	-8328	2976	
20	Итог Бюджет ($)		4632546	4666161	4603712	13902419	4610871	4639557	4621305	13871733	4613753	4619878	4640161	138
21	Итог Выполнено ($)		4617889	4622593	4619293	13859775	4611655	4603912	4650419	13865986	4645645	4620522	4603124	138
22	Итог Разница ($)		14657	43568	-15581	42644	-784	35645	-29114	5747	-31892	-644	37037	
23														

СводнаяТаблица / Лист1

Рис. 17.4. Сводная таблица, которая создана на основе данных, приведенных на рис. 17.3

Код создания сводной таблицы

Ниже, в листинге 17.2, приведен код, который использовался для создания этой сводной таблицы.

Листинг 17.2. Создание структурированной сводной таблицы

```
Sub CreatePivotTable()
    Dim PTcache As PivotCache
    Dim PT As PivotTable

    On Error Resume Next
    Sheets("Лист1").DrawingObjects("TextBoxWait").Visible = True
    On Error GoTo 0
```

```
    Application.ScreenUpdating = False

'   Удаление сводной таблицы, если она существует
    On Error Resume Next
    Application.DisplayAlerts = False
    Sheets("СводнаяТаблица").Delete
    On Error GoTo 0

'   Создание объекта PivotCache
    Set PTcache = ActiveWorkbook.PivotCaches.Add( _
      SourceType:=xlDatabase, _
      SourceData:=Range("A1").CurrentRegion.Address)

'   Добавление рабочего листа
    Worksheets.Add
    ActiveSheet.Name = "СводнаяТаблица"

'   Создание на основе данных кэш-памяти сводной таблицы
    Set PT = PTcache.CreatePivotTable( _
      TableDestination:=Sheets("СводнаяТаблица").Range("A1"), _
      TableName:="BudgetPivot")

    With PT
'       Добавление полей
        .PivotFields("Подразделение").Orientation = xlRowField
        .PivotFields("Месяц").Orientation = xlColumnField
        .PivotFields("Отдел").Orientation = xlPageField
        .PivotFields("Бюджет").Orientation = xlDataField
        .PivotFields("Выполнено").Orientation = xlDataField

'       Добавление вычисляемого поля для учета разницы
        .CalculatedFields.Add "Разница", "=Бюджет-Выполнено"
        .PivotFields("Разница").Orientation = xlDataField

'       Добавление вычисляемых элементов
        .PivotFields("Месяц").CalculatedItems.Add "Кв1", "= Январь+ Февраль+ Март"
        .PivotFields("Месяц").CalculatedItems.Add "Кв2", "= Апрель+ Май+ Июнь"
        .PivotFields("Месяц").CalculatedItems.Add "Кв3", "= Июль+ Август+ Сентябрь"
        .PivotFields("Месяц").CalculatedItems.Add "Кв4", "= Октябрь+ Ноябрь+ Декабрь"

'       Перемещение вычисляемых элементов
        .PivotFields("Месяц").PivotItems("Кв1").Position = 4
        .PivotFields("Месяц").PivotItems("Кв2").Position = 8
        .PivotFields("Месяц").PivotItems("Кв3").Position = 12
        .PivotFields("Месяц").PivotItems("Кв4").Position = 16

'       Изменение подписей
        .PivotFields("Сумма по полю Бюджет").Caption = "Бюджет ($)"
        .PivotFields("Сумма по полю Выполнено").Caption = "Выполнено ($)"
        .PivotFields("Сумма по полю Разница").Caption = "Разница ($)"

    End With
    Application.ScreenUpdating = True
    On Error Resume Next
    Sheets("Лист1").DrawingObjects("TextBoxWait").Visible = False
End Sub
```

Как это работает

Вторая процедура `CreatePivotTable` (листинг 17.2) начинает свою работу с удаления листов `PivotSheet`, если они существуют. После этого создается объект `PivotCache`, добавляется новый лист `СводнаяТаблица`, и создается сводная таблица. Далее программа добавляет поля к созданной сводной таблице.

- Подразделение — Поле строки.
- Месяц — Поле столбца.
- Отдел — Поле страницы.
- Бюджет — Поле данных.
- Выполнено — Поле данных.

Процедура использует метод Add коллекции CalculatedFields для создания вычислимого поля Разница, которое отнимает значение поля Выполнено от значения поля Бюджет. В коде суммируются четыре вычисляемых значения для получения квартальных сумм. По умолчанию вычисляемые значения добавляются в правой части сводной таблицы. Нам же необходимо разместить эти значения возле месяцев, к которым эти значения относятся (например, значение Кв1 размещается возле месяца Март). Наконец, в коде изменяются подписи, которые отображаются в сводной таблице. Например, Сумма по полю Бюджет будет заменена на Бюджет ($).

НА ЗАМЕТКУ

Эта процедура была получена в результате записи действий по созданию и изменению сводной таблицы. После этого код очищается, чтобы стать более понятным и эффективным.

Создание сводной таблицы на основе внешней базы данных

В предыдущих примерах источником данных для сводной таблицы служил рабочий лист. Известно, что Excel позволяет использовать внешний источник данных для создания сводной таблицы. Пример в этом разделе демонстрирует способ создания кода VBA, который формирует сводную таблицу на основе данных, содержащихся в файле базы данных Access.

НА ЗАМЕТКУ

База данных состоит из единой таблицы, которая по своему содержимому идентична таблице, показанной в предыдущем примере.

Код, который создает сводную таблицу в этом случае, показан в листинге 17.3. Он предполагает, что файл базы данных budget.mdb хранится в той же папке, что и рабочая книга.

Листинг 17.3. Создание сводной таблицы на основе информации внешней базы данных

```
Sub CreatePivotTableFromDB()
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim DBFile As String
    Dim ConString As String
    Dim QueryString As String

'   Удаление сводной таблицы, если она существует
    On Error Resume Next
    Application.DisplayAlerts = False
    Sheets("СводнаяТаблица").Delete
    On Error GoTo 0
    Application.DisplayAlerts = True
```

```
'   Создание объекта PivotCache
    Set PTCache = ActiveWorkbook.PivotCaches.Add _
      (SourceType:=xlExternal)

'   Подключение к базе данных и создание запроса
    DBFile = ThisWorkbook.Path & "\budget.mdb"

    ConString = "ODBC;DSN=MS Access Database;DBQ=" & DBFile

    QueryString = "SELECT * FROM BUDGET"

    With PTCache
        .Connection = ConString
        .CommandText = QueryString
    End With

'   Добавление рабочего листа
    Worksheets.Add
    ActiveSheet.Name = "СводнаяТаблица"

'   Создание сводной таблицы
    Set PT = PTCache.CreatePivotTable( _
      TableDestination:=Sheets("СводнаяТаблица").Range("A1"), _
      TableName:="BudgetPivot")

'   Добавление полей
    With PT
        .PivotFields("Подразделение").Orientation = xlRowField
        .PivotFields("Месяц").Orientation = xlColumnField
        .PivotFields("Отдел").Orientation = xlPageField
        .PivotFields("Бюджет").Orientation = xlDataField
        .PivotFields("Выполнено").Orientation = xlDataField
    End With
End Sub
```

Обратите внимание, что аргумент `SourceType` метода `Add` коллекции `PivotCaches` указывается как `xlExternal`. В примере предыдущего раздела (в котором использовались данные рабочего листа) аргумент `SourceType` был равен `xlDatabase`.

Объект `PivotCache` нуждается в получении следующей информации из внешнего файла.

- *Строка соединения.* Данная строка описывает источник данных и имя файла. В этом примере строка соединения указывает источник данных ODBC, который является файлом Microsoft Access — `budget.mdb`.
- *Строка запроса.* Это оператор языка SQL, который определяет, какие записи и поля будут получены из базы данных. В данном примере выбирается вся таблица `Budget`.

Данная информация передается объекту `PivotCache` в результате определения свойств `Connection` и `CommandText`. Как только данные будут сохранены в кэше, с помощью метода `CreatePivotTable` создается сводная таблица.

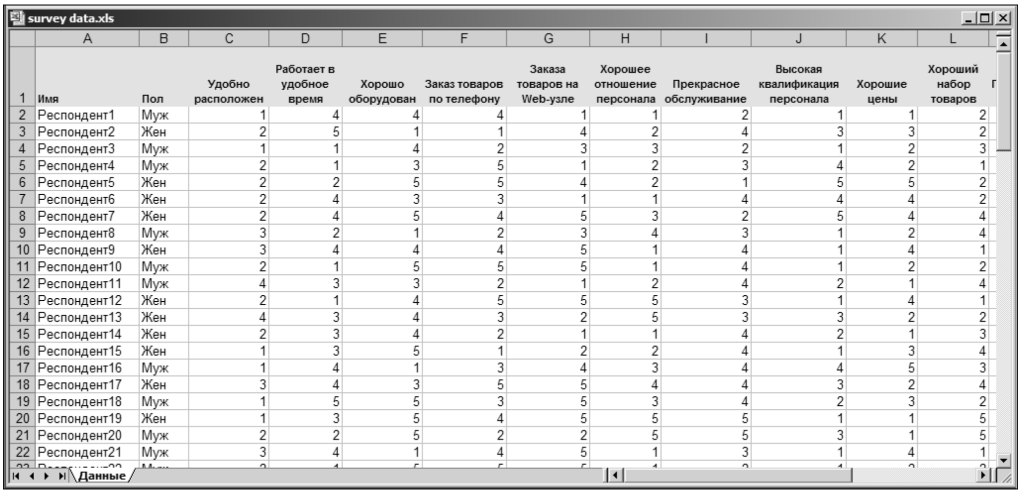
НА ЗАМЕТКУ

SQL является стандартным языком создания запросов к базе данных. Для получения подробной информации вам стоит обратиться к справочному руководству. Советуем также приобрести книгу, которая посвящена программированию на языке SQL.

Создание нескольких сводных таблиц

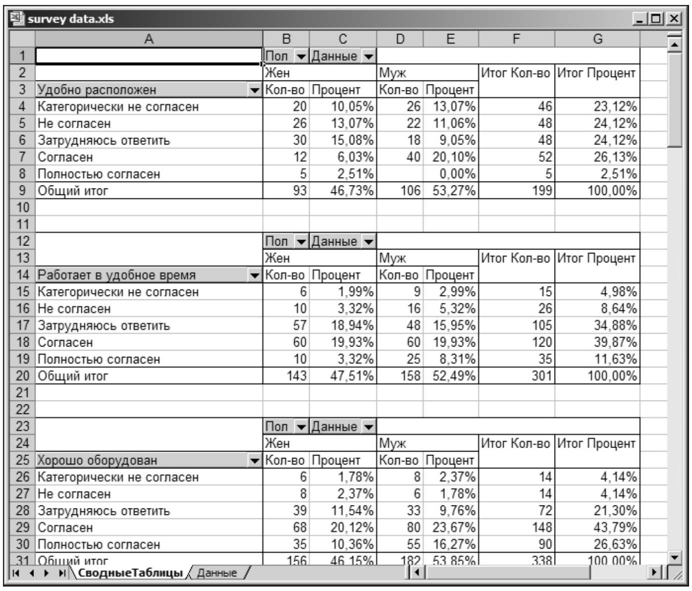
В последнем примере мы создадим набор сводных таблиц, который описывает данные, полученные в результате опроса потребителей. Эти данные хранятся в базе данных на рабочем листе (рис. 17.5) и состоят из 100 строк. Каждая строка содержит информацию о поле потребителя, а также выставленную им по пятибалльной шкале оценку каждому утверждению опроса.

На рис. 17.6 показаны некоторые сводные таблицы, которые получены на основе данных опроса. Каждая сводная таблица предоставляет информацию о распределении оценок для всех утверждений среди респондентов обоих полов.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Имя	Пол	Удобно расположен	Работает в удобное время	Хорошо оборудован	Заказ товаров по телефону	Заказа товаров на Web-узле	Хорошее отношение персонала	Прекрасное обслуживание	Высокая квалификация персонала	Хорошие цены	Хороший набор товаров
2	Респондент1	Муж	1	4	4	4	1	1	2	1	1	2
3	Респондент2	Жен	2	5	1	1	4	2	4	3	3	2
4	Респондент3	Муж	1	1	4	2	3	3	2	1	2	3
5	Респондент4	Муж	2	1	3	5	1	2	3	4	2	1
6	Респондент5	Жен	2	2	5	5	4	2	1	5	5	2
7	Респондент6	Жен	2	4	3	3	1	1	4	4	4	2
8	Респондент7	Жен	2	4	5	4	5	3	2	5	4	4
9	Респондент8	Муж	3	2	1	2	3	4	3	1	2	4
10	Респондент9	Жен	3	4	4	4	5	1	4	1	4	1
11	Респондент10	Муж	2	1	5	5	5	1	4	1	2	2
12	Респондент11	Муж	4	3	3	2	1	2	4	2	1	4
13	Респондент12	Жен	2	1	4	5	5	5	3	1	4	1
14	Респондент13	Жен	4	3	4	3	2	5	3	3	2	2
15	Респондент14	Жен	2	3	4	2	1	1	4	2	1	3
16	Респондент15	Жен	1	3	5	1	2	2	4	1	3	4
17	Респондент16	Муж	1	4	1	3	4	3	4	4	5	3
18	Респондент17	Жен	3	4	3	5	5	4	4	3	2	4
19	Респондент18	Муж	1	5	5	3	5	3	4	2	3	2
20	Респондент19	Жен	1	3	5	4	5	5	5	1	1	5
21	Респондент20	Муж	2	2	5	2	2	5	5	3	1	5
22	Респондент21	Муж	3	4	1	4	5	1	3	1	4	1

Рис. 17.5. Создание набора сводных таблиц позволяет представить данные опроса в более понятном виде



	A	B	C	D	E	F	G
1		Пол	Данные				
2		Жен		Муж		Итог Кол-во	Итог Процент
3	Удобно расположен	Кол-во	Процент	Кол-во	Процент		
4	Категорически не согласен	20	10,05%	26	13,07%	46	23,12%
5	Не согласен	26	13,07%	22	11,06%	48	24,12%
6	Затрудняюсь ответить	30	15,08%	18	9,05%	48	24,12%
7	Согласен	12	6,03%	40	20,10%	52	26,13%
8	Полностью согласен	5	2,51%		0,00%	5	2,51%
9	Общий итог	93	46,73%	106	53,27%	199	100,00%
10							
11							
12		Пол	Данные				
13		Жен		Муж		Итог Кол-во	Итог Процент
14	Работает в удобное время	Кол-во	Процент	Кол-во	Процент		
15	Категорически не согласен	6	1,99%	9	2,99%	15	4,98%
16	Не согласен	10	3,32%	16	5,32%	26	8,64%
17	Затрудняюсь ответить	57	18,94%	48	15,95%	105	34,88%
18	Согласен	60	19,93%	60	19,93%	120	39,87%
19	Полностью согласен	10	3,32%	25	8,31%	35	11,63%
20	Общий итог	143	47,51%	158	52,49%	301	100,00%
21							
22							
23		Пол	Данные				
24		Жен		Муж		Итог Кол-во	Итог Процент
25	Хорошо оборудован	Кол-во	Процент	Кол-во	Процент		
26	Категорически не согласен	6	1,78%	8	2,37%	14	4,14%
27	Не согласен	8	2,37%	6	1,78%	14	4,14%
28	Затрудняюсь ответить	39	11,54%	33	9,76%	72	21,30%
29	Согласен	68	20,12%	80	23,67%	148	43,79%
30	Полностью согласен	35	10,36%	55	16,27%	90	26,63%
31	Общий итог	156	46,15%	182	53,85%	338	100,00%

Рис. 17.6. Эти сводные таблицы созданы с помощью процедуры, написанной на VBA

Код VBA, который позволил создать эти сводные таблицы, показан в листинге 17.4.

Листинг 17.4. Создание нескольких сводных таблиц на основе сложной внешней базы данных

```
Sub MakePivotTables()
'   Процедура создает 14 сводных таблиц
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim SummarySheet As Worksheet
    Dim ItemName As String
    Dim Row As Integer, i As Integer

    Application.ScreenUpdating = False

'   Удаление листа сводных таблиц, если он существует
    On Error Resume Next
    Application.DisplayAlerts = False
    Sheets("СводныеТаблицы").Delete
    On Error GoTo 0

'   Добавление листа сводных таблиц
    Set SummarySheet = Worksheets.Add
    ActiveSheet.Name = "СводныеТаблицы"

'   Создание объекта PivotCache
    Set PTCache = ActiveWorkbook.PivotCaches.Add( _
      SourceType:=xlDatabase, _
      SourceData:=Sheets("Данные").Range("A1"). _
        CurrentRegion.Address)

    Row = 1
    For i = 1 To 14
        ItemName = Sheets("Данные").Cells(1, i + 2)
'       Создание сводной таблицы
        Set PT = PTCache.CreatePivotTable _
            (TableDestination:=SummarySheet.Cells(Row, 1), _
             TableName:=ItemName)
        Row = Row + 11

'       Добавление полей
        With PT.PivotFields(ItemName)
            .Orientation = xlDataField
            .Name = "Кол-во"
        End With

        With PT.PivotFields(ItemName)
            .Orientation = xlDataField
            .Name = "Процент"
            .Calculation = xlPercentOfTotal
        End With

        PT.AddFields RowFields:=Array(ItemName, "Данные")
        PT.PivotFields("Пол").Orientation = xlColumnField
        PT.PivotFields("Данные").Orientation = xlColumnField
     Next i

'   Замена оценок описательным текстом
    SummarySheet.Activate
    With Columns("A:A")
        .Replace "1", "Категорически не согласен"
        .Replace "2", "Не согласен"
        .Replace "3", "Затрудняюсь ответить"
        .Replace "4", "Согласен"
        .Replace "5", "Полностью согласен"
    End With
```

```
'   Настройка ширины столбцов
    Columns("A:G").EntireColumn.AutoFit
End Sub
```

Обратите внимание, что сводные таблицы создаются в цикле на основе одного и того же объекта `PivotCache`. Переменная `Row` отслеживает начало каждой сводной таблицы. После создания сводных таблиц код заменяет числовые категории на текст в первом столбце таблицы (например, 1 заменяется на `Полностью согласен`). Наконец, производится изменение ширины столбцов.

Модификация сводных таблиц

Цель сводных таблиц Excel — предоставить максимальную гибкость при отображении данных. Например, пользователи легко могут менять поле строки на поле столбца и скрывать некоторые элементы сводной таблицы, которые не являются важными в данный момент. Вы вправе предоставить собственный интерфейс, который позволит облегчить пользователям выполнение операций изменения сводных таблиц. Пример, приведенный в этом разделе, представляет сводную таблицу, которой управляет набор элементов `OptionButtons` и два элемента `CheckBox` (рис. 17.7).

Сводная таблица состоит из четырех дополнительных вычисляемых значений (`Кв1`, `Кв2`, `Кв3` и `Кв4`), которые содержат квартальные суммы. Код VBA, который выполняется при щелчке на переключателе `OptionButton1`, показан в листинге 17.5. Данная процедура проста. Она напоминает процедуры обработки событий остальных элементов управления `OptionButton`.



Рис. 17.7. Пользователь может применять элементы управления для изменения параметров сводной таблицы

Листинг 17.5. Ответ на изменение параметров сводной таблицы

```
Private Sub OptionButton2_Click()
'   Только по кварталам
    Application.ScreenUpdating = False
    With ActiveSheet.PivotTables(1).PivotFields("Месяц")
        .PivotItems("Кв1").Visible = True
        .PivotItems("Кв2").Visible = True
        .PivotItems("Кв3").Visible = True
        .PivotItems("Кв4").Visible = True
        .PivotItems("Январь").Visible = False
        .PivotItems("Февраль").Visible = False
        .PivotItems("Март").Visible = False
        .PivotItems("Апрель").Visible = False
```

```
        .PivotItems("Май").Visible = False
        .PivotItems("Июнь").Visible = False
        .PivotItems("Июль").Visible = False
        .PivotItems("Август").Visible = False
        .PivotItems("Сентябрь").Visible = False
        .PivotItems("Октябрь").Visible = False
        .PivotItems("Ноябрь").Visible = False
        .PivotItems("Декабрь").Visible = False
    End With
End Sub
```

Элемент управления CheckBox дает установку отображать общую сумму. Ниже приведены процедуры обработки событий для этих элементов управления.

```
Private Sub CheckBox1_Click()
'   Итоги по столбцам
    Application.ScreenUpdating = False
    ActiveSheet.PivotTables(1).ColumnGrand = CheckBox1.Value
End Sub

Private Sub CheckBox2_Click()
'   Итоги по строкам
    Application.ScreenUpdating = False
    ActiveSheet.PivotTables(1).RowGrand = CheckBox2.Value
End Sub
```

Сводные таблицы могут изменяться и другими способами. Как отмечалось ранее, самый простой способ создания кода VBA, который вносит изменения в сводную таблицу, — это запись макроса при ручном внесении изменений в сводную таблицу. После этого необходимо внести изменения в код и скопировать его в процедуру обработки события элементов управления.

Глава 18

Управление диаграммами

В ЭТОЙ ГЛАВЕ...

Необходимо особо отметить способность Excel к созданию диаграмм. Диаграмма отображает данные любого типа, которые хранятся на рабочем листе. В этой главе рассматриваются следующие вопросы.

- Необходимая информация о диаграммах в Excel.
- Разница между встроенными диаграммами и листами диаграмм.
- Понимание объектной модели `Chart`.
- Использование команды записи макросов для получения информации об объектах `Chart`.
- Примеры выполнения распространенных задач создания диаграмм с помощью VBA.
- Примеры сложных макросов создания диаграмм.
- Интересные приемы создания диаграмм.

Excel поддерживает более 100 различных типов диаграмм, и практически в каждой диаграмме предоставляется возможность управления всеми ее элементами.

О диаграммах

В диаграммах отображается большое количество информации. Они насыщены объектами, каждый из которых имеет собственные свойства и методы. Таким образом, управление диаграммами из кода VBA связано с определенными трудностями. В этой главе рассматриваются основные концепции создания кода VBA, предназначенного для управления диаграммами. Главное — разобраться в тонкостях объектной модели диаграмм. Для начала ознакомимся с общими сведениями о диаграммах Excel.

Расположение диаграмм

В Excel диаграмма может располагаться в нескольких областях рабочей книги.

- В качестве встроенного объекта листа. Лист может содержать любое количество встроенных диаграмм.
- На отдельном листе диаграммы. Такой лист может содержать только одну диаграмму.



Встроенная диаграмма иногда располагается на диалоговом листе, если используется документ Excel 5/95. Как показано далее в этой главе (см. раздел “Сортировка диаграмм на листе диаграммы”), на листе диаграммы можно хранить встроенные диаграммы.

Большинство диаграмм создаются вручную с помощью мастера диаграмм. Их также можно создавать с помощью кода VBA. Кроме того, код VBA используется для модификации уже существующих диаграмм.

Самым быстрым способом создания диаграммы на новом листе является выделение данных и нажатие клавиши <F11>. Excel создаст новый лист диаграммы и воспользуется типом диаграммы, принятым по умолчанию.

Ключевой при работе с диаграммами является концепция "активной диаграммы". Когда пользователь щелкает на встроенной диаграмме или выделяет лист диаграммы, то активизируется объект `Chart`. В VBA свойство `ActiveChart` возвращает активный объект `Chart` (если такой существует). Можно создать код, который будет управлять этим объектом (он будет подобен коду для работы с объектом `Workbook`, возвращаемым свойством `ActiveWorkbook`).

Ниже приведен пример. Если диаграмма активизирована, то следующий оператор отобразит свойство `Name` активного объекта `Chart`.

```
MsgBox ActiveChart.Name
```

Если диаграмма не активизирована, то предыдущий оператор приводит к появлению сообщения об ошибке.

Далее вы узнаете, что необязательно активизировать диаграмму для внесения в нее изменений с помощью кода VBA.

Объектная модель диаграммы

Для того чтобы получить представление о количестве объектов, которые принимают участие в работе диаграммы, воспользуйтесь командой записи макросов, создайте диаграмму и выполните ряд обычных действий по редактированию диаграммы. Полученный код, сгенерированный Excel, может вас удивить своим объемом. При первых попытках изучить объектную модель `Chart` у вас, скорее всего, ничего не получится. В большом количестве объектов легко запутаться, что не удивительно, так как объектная модель диаграммы является довольно запутанной. Более того, объектная модель `Chart` имеет большое количество вложенных уровней.

Предположим, что необходимо изменить заголовок, отображаемый на встроенной диаграмме. Объект верхнего уровня — это объект `Application` (Excel). Объект `Application` содержит объект `Workbook`, в котором находится объект `Worksheet`. Объект `Worksheet` содержит объект `ChartObject`, а в нем расположен объект `Chart`. Объект `Chart` содержит объект `ChartTitle`, а объект `ChartTitle` имеет свойство `Text`, которое представляет объект, отображаемый в заголовке диаграммы.

Ниже в графическом виде представлена иерархия объектов встроенной диаграммы.

```
Application
  Workbook
    Worksheet
      ChartObject
        Chart
          ChartTitle
```

Код VBA должен, конечно же, следовать этой объектной модели. Например, чтобы установить заголовок диаграммы в значение `Ежегодные продажи`, можно создать следующий оператор VBA.

```
WorkSheets("Лист1").ChartObjects(1).Chart.ChartTitle. _
  .Text = "Ежегодные продажи"
```

В этом выражении предполагается, что активная рабочая книга представлена объектом Workbook. Оператор обращается к первому элементу коллекции рабочих листов Лист1. Свойство Chart возвращает объект Chart, а свойство ChartTitle возвращает объект ChartTitle. В самом конце оператора вы обращаетесь к свойству Text.

Для диаграммы на листе диаграммы объектная модель несколько отличается, так как она не содержит объект Worksheet или ChartObject. Например, ниже представлена объектная модель для диаграммы на листе диаграммы.

```
Application
  Workbook
    Chart
      ChartTitle
```

Также можно обратиться к свойству Text объекта ChartTitle.

```
WorkSheets("Лист1").ChartTitle. _
  Text = "Ежегодные продажи"
```

Другими словами, лист диаграммы является объектом Chart и не содержит объект ChartObject. Если попытаться рассмотреть ситуацию в общем, то легко понять, что родительским объектом для встроенной диаграммы является объект ChartObject, а для диаграммы на отдельном листе диаграммы родительским объектом выступает объект Workbook.

Ниже приведены операторы, которые отображают окно сообщения со словом Диаграмма.

```
MsgBox TypeName(Sheets("Лист1").ChartObjects(1).Chart)

Msgbox TypeName(Sheets("Диаграмма1"))
```

НА ЗАМЕТКУ

При создании новой встроенной диаграммы в объектную модель добавляется коллекция ChartObjects, которая содержится в объекте определенного листа (для листа не создается коллекция Charts). При создании листа диаграммы новый объект всего лишь добавляется в коллекции Charts и Sheets определенной рабочей книги.

Запись макроса

Возможно, самым лучшим способом ознакомления с объектной моделью объекта Chart является запись макроса при создании и внесении изменений в диаграммы. Даже если при записи макросов создается большое количество лишнего и неэффективного кода, этот код можно с успехом использовать в ознакомительных целях — для получения информации об объектах, свойствах и методах, о которых необходимо знать при управлении диаграммами с помощью кода VBA.

Команда записи макросов в Excel генерирует код, в котором используется только активная диаграмма, а для представления самого объекта Chart применяется свойство ActiveChart. В Excel не обязательно выделять объект (или активизировать диаграмму) для того, чтобы управлять им (ею) в коде VBA. Кроме того, как отмечалось ранее, при записи макроса создается большое количество лишнего кода. Таким образом, если эффективность кода является одной из первостепенных задач, то вам придется редактировать код, полученный после записи макроса (особенно, если этот код используется для управления диаграммами).

Результат записи макроса

Команда записи макросов была включена при создании простой диаграммы (показанной на рис. 18.1). Данные диаграммы занесены в диапазон A1:F2. После создания диаграммы (но еще в процессе записи макроса) ее немного изменили.

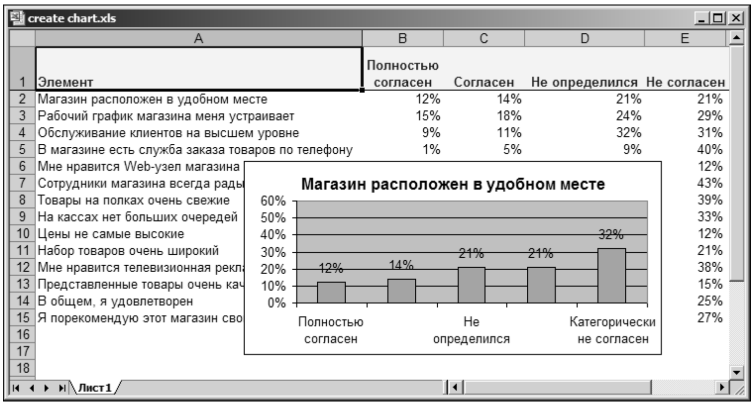


Рис. 18.1. Эта диаграмма создавалась и редактировалась в процессе записи макроса в Excel

Ниже приводится листинг кода, который был получен в результате записи макроса.

```
Sub Macro1()
'   Записанный макрос
    Range("A1:F2").Select
    Charts.Add
    ActiveChart.ChartType = xlColumnClustered
    ActiveChart.SetSourceData _
      Source:=Sheets("Лист1").Range("A1:F2"), _
      PlotBy:=xlRows
    ActiveChart.Location _
      Where:=xlLocationAsObject, _
      Name:="Лист1"
    ActiveChart.HasLegend = False
    ActiveChart.ApplyDataLabels _
      Type:=xlDataLabelsShowValue, LegendKey:=False
    ActiveChart.HasDataTable = False
    ActiveChart.Axes(xlCategory).Select
    Selection.TickLabels.Orientation = xlHorizontal
    ActiveChart.ChartTitle.Select
    Selection.Font.Bold = True
    Selection.AutoScaleFont = True
    With Selection.Font
        .Name = "Arial"
        .Size = 12
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ColorIndex = xlAutomatic
        .Background = xlAutomatic
    End With
    ActiveChart.PlotArea.Select
```

```
    Selection.Top = 18
    Selection.Height = 162
    ActiveChart.ChartArea.Select
    ActiveChart.Axes(xlValue).Select
    With ActiveChart.Axes(xlValue)
        .MinimumScaleIsAuto = True
        .MaximumScale = 0.6
        .MinorUnitIsAuto = True
        .MajorUnitIsAuto = True
        .Crosses = xlAutomatic
        .ReversePlotOrder = False
        .ScaleType = xlLinear
    End With
End Sub
```

Подкорректированный код

Большая часть кода, полученного при записи макроса в предыдущем разделе, не так важна, как кажется на первый взгляд. Она используется для установки значений свойств, которые в дальнейшем использоваться не будут. Ниже приведен листинг отредактированного кода макроса. Этот код выполняет те же действия, что и код из предыдущего раздела, однако он намного короче и эффективнее. Установка свойства `ScreenUpdating` в значение `False` предотвращает операцию обновления экрана.

```
Sub CleanedMacro()
    Application.ScreenUpdating = False
    Charts.Add
    ActiveChart.Location _
      Where:=xlLocationAsObject, Name:="Лист1"
    With ActiveChart
        .SetSourceData Range("A1:F2")
        .HasTitle = True
        .ChartType = xlColumnClustered
        .HasLegend = False
        .ApplyDataLabels Type:=xlDataLabelsShowValue
        .Axes(xlCategory).TickLabels.Orientation = xlHorizontal
        .ChartTitle.Font.Bold = True
        .ChartTitle.Font.Size = 12
        .PlotArea.Top = 18
        .PlotArea.Height = 162
        .Axes(xlValue).MaximumScale = 0.6
        .Deselect
    End With
    Application.ScreenUpdating = True
End Sub
```

НА ЗАМЕТКУ

При создании диаграммы с помощью метода `Add` коллекции `Charts` вы всегда будете использовать лист диаграммы. В предыдущем коде метод `Location` перемещал диаграмму на рабочий лист.

НА КОМПАКТ-ДИСКЕ

Рабочая книга, которая содержит оба макроса (записанный и подкорректированный), находится на прилагаемом к книге компакт-диске. Загрузив эту рабочую книгу, можно самостоятельно сравнить производительность обеих версий макроса.

Распространенные методы управления диаграммами в VBA

В этом разделе описаны способы решения часто возникающих задач, которые выполняются с помощью диаграмм.

Активизация диаграммы

Когда пользователь щелкает на встроенной диаграмме, она активизируется. В VBA можно активизировать встроенную диаграмму с помощью метода `Activate`. Ниже приведен пример применения этого метода.

```
ActiveSheet.ChartObjects("Диаграмма1").Activate
```

Если диаграмма находится на листе диаграммы, то можно воспользоваться следующим оператором.

```
Sheets("Диаграмма1").Activate
```

Как только диаграмма будет активизирована, на нее можно ссылаться с помощью свойства `ActiveChart`. Например, следующий оператор приводит к отображению имени активной диаграммы. Если активной диаграммы в проекте нет, то оператор приведет к возникновению ошибки.

```
MsgBox ActiveChart.Name
```

Чтобы модифицировать диаграмму с помощью кода VBA, ее не обязательно активизировать. Две процедуры, представленные ниже, приводят к одинаковому результату (встроенная диаграмма `Диаграмма1` любого типа превращается в диаграмму с областями). Первая процедура активизирует диаграмму перед выполнением необходимых действий, а вторая выполняет операцию без активизации.

```
Sub ModifyChart1()
    ActiveSheet.ChartObjects("Диаграмма1").Activate
    ActiveChart.Type = xlArea
    ActiveChart.Deselect
End Sub

Sub ModifyChart2()
    ActiveSheet.ChartObjects("Диаграмма1").Chart.Type = xlArea
End Sub
```

Диаграмма, встроенная в рабочий лист, легко преобразуется в лист диаграммы. Для того чтобы выполнить эту операцию вручную, следует активизировать встроенную диаграмму и выполнить команду **Диаграмма⇨Размещение**. В диалоговом окне **Размещение диаграммы** выберите переключатель **отдельном** и укажите имя листа. Это действие приведет к копированию объекта `Chart` (который содержится в объекте `ChartObject`) на лист диаграммы, после чего исходный объект `ChartObject` будет уничтожен.

Кроме того, встроенную диаграмму можно преобразовать в лист диаграммы с помощью кода VBA. Ниже приведен пример такого кода, который преобразует первый объект `ChartObject` на листе `Лист1` в лист диаграммы, который называется `МояДиаграмма`.

```
Sub ConvertChart1()
    Sheets("Лист1").ChartObjects(1).Chart. _
      Location xlLocationAsNewSheet, "МояДиаграмма"
End Sub
```

Следующий пример выполняет действия, противоположные предыдущей процедуре: диаграмма на листе диаграммы `МояДиаграмма` превращается во встроенную диаграмму на листе `Лист1`.

```
Sub ConvertChart2()
    Charts("МояДиаграмма") _
      .Location xlLocationAsObject, "Лист1"
End Sub
```

Использование метода `Location` также приводит к активизации перемещаемой диаграммы.

При активизации диаграмма, представленная объектом `ChartObject`, располагается в окне, которое в обычных условиях остается невидимым. Чтобы увидеть встроенную диаграмму в собственном окне, щелкните правой кнопкой мыши на объекте диаграммы и выберите Окно диаграммы в контекстном меню. Встроенная диаграмма останется на листе, но, кроме этого, диаграмма появится в собственном плавающем окне (рис. 18.2). Данное окно можно перемещать, можно изменять его размер (однако его нельзя разворачивать). Если переместить окно, встроенная диаграмма все равно останется на своем месте. Активизация другого окна приведет к тому, что окно диаграммы вновь станет невидимым.

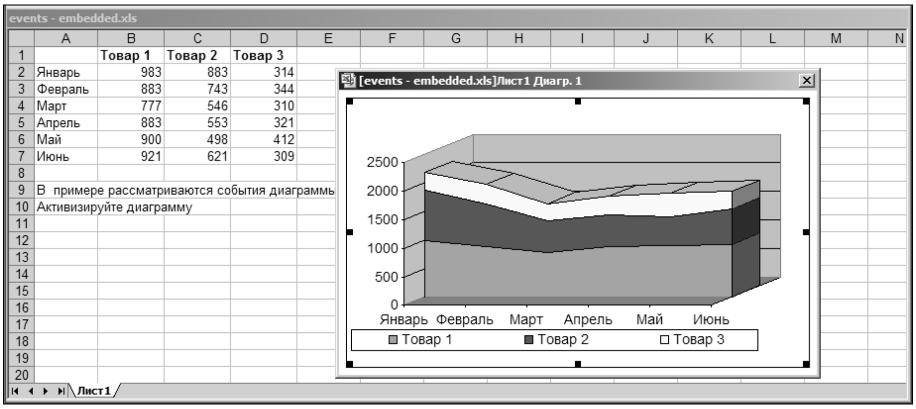


Рис. 18.2. Отображение встроенной диаграммы в собственном окне

Приведенный далее код VBA представляет окно объекта `ChartObject` на активном листе.

```
ActiveSheet.ChartObjects(1).Activate
ActiveChart.ShowWindow = True
```

Чтобы получить информацию о практическом применении окна встроенной диаграммы, обратитесь к разделу “Печать встроенных диаграмм на всю страницу” далее в этой главе.

Деактивизация диаграммы

При записи макроса, который приводит к деактивизации диаграммы, генерируется следующий оператор.

```
ActiveWindow.Visible = False
```

Данный оператор выполняет деактивизацию диаграммы, однако не совсем понятно, почему она происходит. При создании макросов по управлению диаграммами лучше использовать метод Deselect, который также позволяет деактивизировать диаграмму.

```
ActiveChart.Deselect
```

Эти два оператора действуют по-разному. Когда активной является встроенная диаграмма, вызов метода Deselect не приводит к выделению ячеек на листе. Однако установка свойства Visible объекта ActiveWindow в значение False приведет к выделению диапазона, который был выделен перед активизацией диаграммы.

Определение активности диаграммы

Обычно макрос выполняет операции по отношению к активной диаграмме (выделенной пользователем). Например, макрос может изменять тип диаграммы, задавать другие цвета элементов или изменять размер используемого шрифта.

Возникает вопрос: как код VBA определяет, что пользователь выделил диаграмму? Под *выделением диаграммы* подразумевается активизация листа диаграммы или активизация встроенной диаграммы в результате щелчка на ней. Первым этапом является проверка свойства TypeName для объекта Selection с помощью следующего оператора.

```
TypeName(Selection) = "Chart"
```

Это выражение равно значению True, если лист диаграммы активен. Если же выделить встроенную диаграмму, то это выражение *не* будет равно значению True. Кроме того, если выделена встроенная диаграмма, то выделение может представлять объект, который содержится в объекте Chart. Например, выделение может представлять объект Series, объект ChartTitle, объект Legend, объект PlotArea и т.д.

Функция ChartIsSelected, показанная ниже, возвращает значение True, если активным является лист диаграммы или встроенная диаграмма. В противном случае возвращается значение False.

```
Private Function ChartIsSelected() As Boolean
    ChartIsSelected = Not ActiveChart Is Nothing
End Function
```

Данная функция определяет равенство объекта ActiveChart и значения Nothing. Если это так, то активизирована не диаграмма.

Удаление объектов ChartObject или диаграмм

Для того чтобы удалить все объекты ChartObject на рабочем листе, воспользуйтесь методом Delete коллекции ChartObjects.

```
ActiveSheet.ChartObjects.Delete
```

Вы удалите все листы диаграмм в активной рабочей книге, если воспользуетесь таким оператором.

```
ActiveWorkbook.Charts.Delete
```

Обычно при удалении листа в Excel появляется предупреждение, подобное показанному на рис. 18.3. Пользователь должен отреагировать на это предупреждение, чтобы продолжить работу макроса. Вы сможете избавиться от этого предупреждения, если воспользуетесь следующей последовательностью операторов.

```
Application.DisplayAlerts = False
ActiveWorkbook.Charts.Delete
Application.DisplayAlerts = True
```

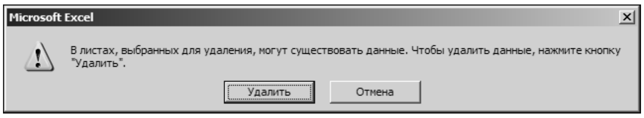


Рис. 18.3. Если вы попытаетесь удалить один или несколько листов диаграмм, то появится это окно сообщения

Форматирование диаграмм

Приведенный далее пример демонстрирует методы форматирования активной диаграммы.

```
Sub ChartMods1()
    With ActiveChart
        .Type = xlArea
        .ChartArea.Font.Name = "Arial"
        .ChartArea.Font.FontStyle = "Regular"
        .ChartArea.Font.Size = 9
        .PlotArea.Interior.ColorIndex = xlNone
        .Axes(xlValue).TickLabels.Font.Bold = True
        .Axes(xlCategory).TickLabels.Font.Bold = True
        .HasLegend = True
        .Legend.Position = xlBottom
    End With
End Sub
```

Диаграмма должна быть активной, в противном случае выполнение этой процедуры приведет к появлению сообщения об ошибке. Также обратите внимание на то, что в коде свойство `HasLegend` устанавливается в значение `True`. Таким образом, вы сможете избежать сообщения об ошибке при установке свойства `Position` объекта `Legend`, если диаграмма не содержит легенду.

Рабочая книга с этим примером также находится на прилагаемом к книге компакт-диске.

Ниже приведена другая версия процедуры `ChartMods`. В данном случае она применяется к определенной диаграмме — той, которая содержится в объекте `ChartObject` с названием `Диаграмма1` и расположена на листе `Лист1`. Обратите внимание на то, что активизация диаграммы не производится.

```
Sub ChartMods2()
    With Sheets("Лист1").ChartObjects("Диаграмма1").Chart
        .Type = xlArea
        .ChartArea.Font.Name = "Arial"
        .ChartArea.Font.FontStyle = "Regular"
        .ChartArea.Font.Size = 9
        .PlotArea.Interior.ColorIndex = xlNone
        .Axes(xlValue).TickLabels.Font.Bold = True
        .Axes(xlCategory).TickLabels.Font.Bold = True
        .HasLegend = True
        .Legend.Position = xlBottom
    End With
End Sub
```

Циклический просмотр диаграмм

В некоторых случаях необходимо выполнить операцию над всеми диаграммами. Приведенный ниже пример изменяет тип каждой встроенной диаграммы на активном листе. Процедура использует цикл `For Next` для циклического просмотра объектов в коллекции `ChartObjects`. После этого изменяется свойство `ChartType` каждого объекта `Chart`. Диаграмма с областями назначается с помощью предопределенной константы `xlArea`. Обратитесь к справочному руководству по VBA, чтобы получить информацию о константах, определяющих остальные типы диаграмм.

```
Sub ChangeChartType()
    Dim chtobj as ChartObject
    For Each chtobj In ActiveSheet.ChartObjects
        chtobj.Chart.ChartType = xlArea
    Next chtobj
End Sub
```

Следующий макрос выполняет ту же операцию, что и предыдущий, но работает со всеми листами диаграмм в активной рабочей книге.

```
Sub ChangeChartType2()
    Dim cht as Chart
    For Each cht In ActiveWorkbook.Charts
        cht.ChartType = xlArea
    Next cht
End Sub
```

Приведенный далее пример изменяет шрифт текста легенды для всех диаграмм на активном листе. Для просмотра объектов `ChartObject` используется цикл `For Next`.

```
Sub LegendMod()
    Dim chtobj as ChartObject
    For Each chtobj In ActiveSheet.ChartObjects
        With chtobj.Chart.Legend.Font
            .Name = "Arial"
            .FontStyle = "Bold"
            .Size = 12
        End With
    Next chtobj
End Sub
```

Изменение размера и взаимного расположения объектов ChartObject

Объект `ChartObject` имеет стандартные свойства. Они задают расположение и размер, к которым можно получать доступ с помощью VBA. Представленный ниже пример демонстрирует изменение размера всех объектов `ChartObject` на листе `Лист1`, в результате чего они будут соответствовать размерам объекта `ChartObject` с названием `Диаграмма1`. Кроме того, взаимное расположение объектов `ChartObject` устанавливается таким образом, чтобы они размещались один над другим в левой части листа.

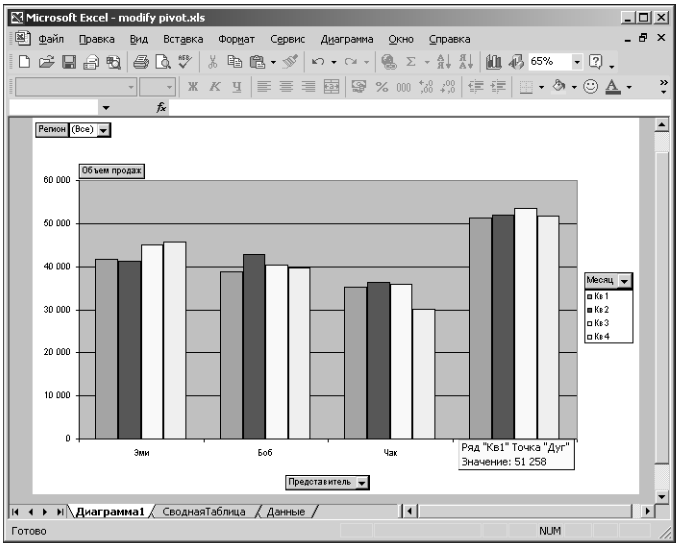
```
Sub ResizeAndArrangeChartObjects()
    Dim W As Double, H As Double
    Dim TopPos As Double
    Dim chtobj As ChartObject

    W = ActiveSheet.ChartObjects("Диаграмма1").Width
    H = ActiveSheet.ChartObjects("Диаграмма1").Height
    TopPos = 0
```

```
    For Each chtobj In ActiveSheet.ChartObjects
        With chtobj
            .Width = W
            .Height = H
            .Left = 0
            .Top = TopPos
        End With
        TopPos = TopPos + H
    Next chtobj
End Sub
```

Работа со сводными диаграммами

В Excel 2000 был представлен новый вид диаграмм — сводная диаграмма (объект `PivotChart`). Это средство позволяет создавать динамические диаграммы, которые привязываются к сводной таблице (объект `PivotTable`). Диаграмма `PivotChart` графически отображает текущую структуру сводной таблицы (`PivotTable`). При создании сводной таблицы предоставляется возможность создания сводной диаграммы, которая будет содержать связанную с ней структуру данных сводной таблицы. Для создания сводной диаграммы на основе существующей сводной таблицы активизируйте сводную таблицу и щелкните на кнопке запуска мастера диаграмм. На новом листе диаграммы будет создана сводная диаграмма. По умолчанию новая сводная диаграмма всегда находится на листе диаграммы (см. рисунок). Однако можно воспользоваться командой Диаграмма⇨Расположение, чтобы преобразовать диаграмму на отдельном листе во встроенную диаграмму.



Когда Microsoft добавляет новую возможность в Excel, приходится вносить изменения в объектную модель Excel, чтобы к новой возможности можно было получить доступ с помощью VBA. В сводных диаграммах объект `PivotLayout` находится в объекте `Chart`. Наиболее удачным способом ознакомления с возможностями этого объекта является запись макросов при ручном создании и модификации сводных диаграмм. Дополнительную информацию об объектах, свойствах и методах можно получить, обратившись к справочному руководству по VBA.

Переменная TopPos отслеживает вертикальное расположение следующей диаграммы. Каждый раз при переходе к следующей диаграмме (следующая итерация) эта переменная увеличивается на значение H (которое равно высоте объекта ChartObject).

Рабочая книга с этим примером также находится на прилагаемом к книге компакт-диске.

Дополнительные методы управления диаграммами

В этом разделе рассматриваются дополнительные методы работы с диаграммами. Вы будете иметь возможность ознакомиться с примерами, которые демонстрируют использование VBA для изменения данных, используемых диаграммой.

Использование имен в формуле РЯД

Диаграмма может состоять из любого количества последовательностей. Данные, которые используются каждой последовательностью, определяются ссылками на диапазоны в формуле РЯД (SERIES). Дополнительная информация по этой теме приводится во врезке "Формула РЯД в диаграмме".

В некоторых случаях использование имен диапазонов в формуле РЯД диаграммы может намного упростить ситуацию, особенно если необходимо изменить источник данных диаграммы с помощью кода VBA. В качестве примера приведем следующую формулу РЯД.

```
=РЯД(;Лист1!$A$1:$A$6;Лист1!$B$1:$B$6;1)
```

Формула РЯД в диаграмме

Данные, которые используются в каждой последовательности диаграммы, определяются формулой РЯД. При выделении последовательностей данных на диаграмме формула РЯД отображается в строке формул. Это не "настоящая" формула: ее нельзя использовать в ячейке рабочего листа, кроме того, вы не сможете применить в этой формуле функции. Однако допускается редактировать аргументы формулы РЯД.
Формула РЯД имеет следующий синтаксис.

```
=РЯД(имя; подписи_категорий; значения;, порядок)
```

- имя — указывает имя, которое используется в легенде (необязательный параметр). Если на диаграмме присутствует только одна последовательность, то значение этого аргумента применяется в качестве заголовка;
- подписи_категорий — указывает диапазон, который состоит из подписей для шкалы категорий (необязательный параметр). Если его пропустить, то Excel будет использовать последовательность целых чисел, начиная с 1.
- значения — указывает диапазон, который содержит значения последовательности.
- порядок — это целое число, которое указывает порядок представления последовательностей (используется лишь тогда, когда диаграмма содержит более одной последовательности).

Ссылки на диапазоны в формуле `РЯД` всегда абсолютные и содержат имя листа.

```
=РЯД(Лист!$B$1;;Лист!$B$2:$B$7;1)
```

Ссылка может задаваться на несмежный диапазон. Если это так, то каждый поддиапазон разделяется точкой с запятой, а аргумент заключается в скобки. В следующей формуле в качестве значения указан диапазон `B2:B3` и `B5:B7`.

```
=РЯД(;;(Лист1!$B$2:$B$3;Лист1!$B$5:$B$7);1)
```

Вместо ссылок на диапазоны можно вставить их имена. В результате Excel изменит ссылку в формуле `РЯД` так, чтобы в ней использовалось имя рабочей книги.

```
=РЯД(Лист1!$B$1;;budget.xls!MyData;1)
```

Можно определить имена для двух диапазонов (например, `Категории` и `Данные`), после чего отредактировать формулу `РЯД` таким образом, чтобы она содержала имена диапазонов вместо ссылок на эти диапазоны. Отредактированная формула будет выглядеть следующим образом.

```
=РЯД(;Лист!Категории;Лист!Данные;1)
```

Как только имена будут определены, а формула `РЯД` — отредактирована, в коде VBA можно использовать имена. Все проведенные изменения будут отображены на диаграмме. Например, приведенный ниже оператор "заменяет" ссылку на диапазон именем `Данные`.

```
Range("B1:B12").Name = "Данные"
```

После выполнения этого оператора диаграмма обновляется и использует новое определение диапазона.

СОВЕТ

Метод `Resize` объекта `Range` рекомендуется использовать при изменении размера именованного диапазона. Например, следующий код расширяет диапазон `Data` так, что он пополняется новой строкой.

```
With Range("Data")
    .Resize(.Rows.Count + 1, 1).Name = "Data"
End With
```

НОВИНКА

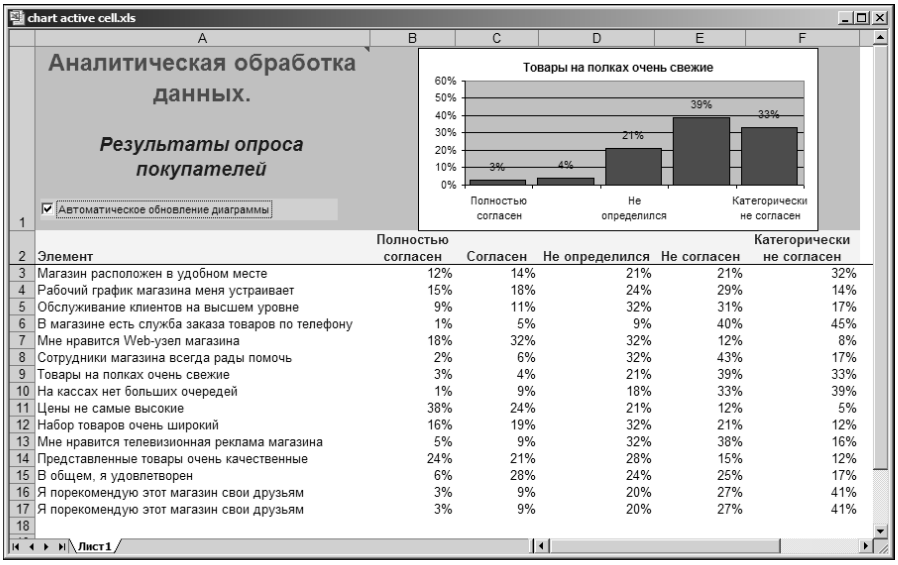
В Excel 2003 появилось новое средство, с помощью которого диаграмма может быть построена на основе расширяемого диапазона данных. Воспользовавшись командой Данные⇨Список⇨Создать список, вы сможете добавлять в диапазон новые данные, которые будут немедленно представляться на диаграмме.

Определение данных, которые используются диаграммой

Примеры, приведенные в этом разделе, демонстрируют способы использования VBA для изменения тех данных, на основе которых создается диаграмма.

ВЫБОР ДАННЫХ ДЛЯ ДИАГРАММЫ НА ОСНОВЕ АКТИВНОЙ ЯЧЕЙКИ

На рис. 18.4 показана диаграмма, которая основана на данных строки активной ячейки. При активизации пользователем другой ячейки диаграмма автоматически обновляется.



	Элемент	Полностью согласен	Согласен	Не определился	Не согласен	Категорически не согласен
3	Магазин расположен в удобном месте	12%	14%	21%	21%	32%
4	Рабочий график магазина меня устраивает	15%	18%	24%	29%	14%
5	Обслуживание клиентов на высшем уровне	9%	11%	32%	31%	17%
6	В магазине есть служба заказа товаров по телефону	1%	5%	9%	40%	45%
7	Мне нравится Web-узел магазина	18%	32%	32%	12%	8%
8	Сотрудники магазина всегда рады помочь	2%	6%	32%	43%	17%
9	Товары на полках очень свежие	3%	4%	21%	39%	33%
10	На кассах нет больших очередей	1%	9%	18%	33%	39%
11	Цены не самые высокие	38%	24%	21%	12%	5%
12	Набор товаров очень широкий	16%	19%	32%	21%	12%
13	Мне нравится телевизионная реклама магазина	5%	9%	32%	38%	16%
14	Представленные товары очень качественные	24%	21%	28%	15%	12%
15	В общем, я удовлетворен	6%	28%	24%	25%	17%
16	Я порекомендую этот магазин свои друзьям	3%	9%	20%	27%	41%
17	Я порекомендую этот магазин свои друзьям	3%	9%	20%	27%	41%

Рис. 18.4. Эта диаграмма всегда отображает данные строки активной ячейки

В этом примере использована процедура обработки события для объекта `Worksheet`. Событие `SelectionChange` происходит каждый раз, когда пользователь изменяет активную ячейку, тем самым влияя на выделение. Процедура обработки этого события (которая находится в модуле кода для объекта `Лист1`) приводится ниже.

```
Private Sub Worksheet_SelectionChange(ByVal Target _
 As Excel.Range)
    Call UpdateChart
End Sub
```

Другими словами, каждый раз, когда пользователь изменяет активную ячейку, выполняется процедура `Worksheet_SelectionChange`. Эта процедура вызывает процедуру `UpdateChart`.

```
Sub UpdateChart()
    Dim TheChartObj As ChartObject
    Dim TheChart As Chart
    Dim UserRow As Long
    Dim CatTitles As Range
    Dim SrcRange As Range
    Dim SourceData As Range

    If Sheets("Лист1").CheckBox1 Then
        Set TheChartObj = ActiveSheet.ChartObjects(1)
        Set TheChart = TheChartObj.Chart
        UserRow = ActiveCell.Row
        If UserRow < 3 Or IsEmpty(Cells(UserRow, 1)) Then
            TheChartObj.Visible = False
        Else
            Set CatTitles = Range("A2:F2")
            Set SrcRange = Range(Cells(UserRow, 1), Cells(UserRow, 6))
            Set SourceData = Union(CatTitles, SrcRange)
            TheChart.SetSourceData _
              Source:=SourceData, PlotBy:=xlRows
            TheChartObj.Visible = True
        End If
    End If
End Sub
```

Первым шагом процедуры является определение состояния флажка Автоматическое обновление диаграммы. Если этот флажок не установлен, то ничего не происходит. Переменная `UserRow` содержит номер строки активной ячейки. Оператор `If` проверяет расположение активной ячейки в строке, содержащей данные (данные начинаются со строки 3). Если указатель на ячейку находится в строке, не содержащей данные, то объект `ChartObject` скрывается. В противном случае код создает объект `Range` (с именем `CatTitle`), который содержит подписи категорий, и другой объект `Range` (с именем `SrcRange`), который содержит данные строки. Эти два объекта `Range` объединяются с помощью функции VBA `Union` и присваиваются объекту `Range` под именем `SourceData`. Наконец, диапазон `SourceData` указывается в качестве данных диаграммы посредством метода `SetSourceData` объекта `Chart`.

Изменение диаграммы с помощью элемента управления ComboBox

Следующий пример демонстрирует использование элемента управления `ComboBox` для управления листом диаграммы. Этот элемент управления позволяет выбрать тип диаграммы в раскрывающемся списке (рис. 18.5).

Элемент управления `ComboBox`, который использовался в приведенном примере, вставлен с панели инструментов Формы (а не с панели инструментов Элементы управления). Помните, что Excel не позволяет добавлять на лист диаграммы элементы управления ActiveX.

Эта рабочая книга доступна на прилагаемом к книге компакт-диске.

Макрос `DropDown1_Change` назначается элементу управления `ComboBox`. Когда пользователь выбирает одну из опций в элементе управления `ComboBox`, запускается следующая процедура.

```
Sub DropDown1_Change()
    ListIndex = Charts(1).DropDowns(1).Value
    Call UpdateChart(ListIndex)
End Sub
```

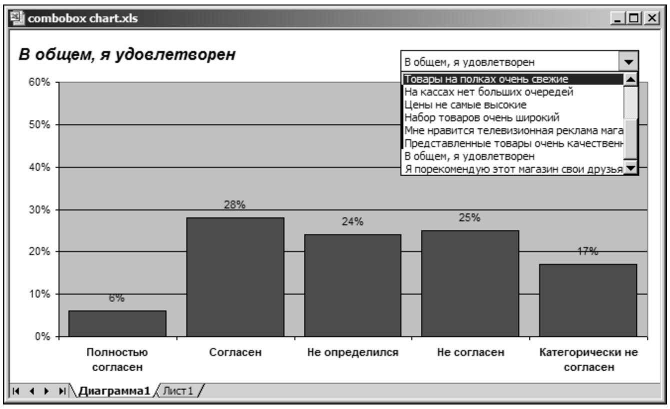


Рис. 18.5. Изменение значения элемента управления `ComboBox` *приводит к замене источника данных диаграммы*

Данная процедура вызывает процедуру UpdateChart и передает ей целое число, которое представляет элемент, выбранный пользователем. Ниже приведен листинг процедуры UpdateChart (она подобна процедуре UpdateChart из предыдущего раздела).

```
Sub UpdateChart(Item)
'   Обновляет диаграмму после выбора опции в списке
    Dim TheChart As Chart
    Dim DataSheet As Worksheet
    Dim CatTitles As Range, SrcRange As Range
    Dim SourceData As Range

    Set TheChart = Sheets("Диаграмма1")
    Set DataSheet = Sheets("Лист1")

    With DataSheet
        Set CatTitles = .Range("A2:F2")
        Set SrcRange = .Range(.Cells(Item + 2, 1), _
          .Cells(Item + 2, 6))
    End With
    Set SourceData = Union(CatTitles, SrcRange)

    With TheChart
        .SetSourceData Source:=SourceData, PlotBy:=xlRows
        .ChartTitle.Left = TheChart.ChartArea.Left
        .Deselect
    End With
End Sub
```

Определение источника данных

Обратимся к объектной модели диаграммы. Объект Series является дочерним по отношению в объекту Chart. Элемент SeriesCollection представляет коллекцию объектов Series для определенного объекта Chart. Если диаграмма отображает две последовательности данных, в этой коллекции содержится два объекта. Потому ссылаться на определенный объект Series можно по индексу. Объект Series наделен несколькими свойствами, но в данном случае нас интересуют только три из них.

- Formula. Возвращает или задает формулу РЯД для объекта Series. Когда выделяется последовательность диаграммы, формула РЯД отображается на панели формул. Свойство Formula возвращает формулу в виде текстовой строки.
- Values. Возвращает или задает коллекцию всех значений в последовательности. Это может быть диапазон на рабочем листе или массив постоянных значений, но не их комбинация.
- XValues. Возвращает или устанавливает массив значений вдоль оси X для последовательности на диаграмме. Свойство XValues может содержать диапазон на рабочем листе или массив значений, но не их комбинацию.

Таким образом, если программный код должен оперировать диапазоном данных, который представлен конкретной последовательностью на диаграмме, понятно, что необходимо применить свойство Values объекта Series. Свойство XValues также используется для получения диапазона, содержащего значения вдоль оси X (или подписи категорий). Теоретически синтаксис правильный, но на практике он не выполняется. При назначении диапазона с помощью свойства XValues происходит автоматическая установка значений Value для каждой ячейки в диапазоне (т.е. массива значений).

Одно из решений — написание кода, который анализирует формулу РЯД (SERIES) и извлекает из нее адреса диапазонов. Реализуется этот процесс довольно сложно, поскольку сама формула РЯД (SERIES) имеет неоднозначный синтаксис.

В качестве аргументов формулы РЯД (SERIES) используются массивы, непрерывные диапазоны, пустые ячейки, кроме того, отдельные аргументы являются необязательными. Все это затрудняет синтаксический анализ формулы.

Я работал над этой проблемой несколько лет, и, в конце концов, мне удалось получить правильное решение задачи. Я написал функцию, которая принимает все аргументы формулы РЯД (SERIES) и возвращает массив 2×5 элементов, которые отображают полные сведения о формуле.

Пришлось разделить сложную функцию на четыре более простые функции, каждая из которых принимает один аргумент (объект Series) и возвращает массив из двух элементов.

- SERIESNAME_FROM_SERIES. Первый элемент массива содержит строку, описывающую тип данных первого аргумента формулы РЯД (SERIES) (Range, Empty или String). Второй элемент массива содержит адрес диапазона, пустую или обычную строку.
- XVALUES_FROM_SERIES. Первый элемент массива содержит строку, описывающую тип данных второго аргумента формулы РЯД (SERIES) (Range, Empty или String). Второй элемент массива содержит адрес диапазона, пустую или обычную строку.
- VALUES_FROM_SERIES. Первый элемент массива содержит строку, описывающую тип данных третьего аргумента формулы РЯД (SERIES) (Range или Array). Второй элемент массива содержит адрес диапазона или массив.
- BUBBLESIZE_FROM_SERIES. XVALUES_FROM_SERIES. Первый элемент массива содержит строку, описывающую тип данных пятого аргумента формулы РЯД (SERIES) (Range, Array или Empty). Второй элемент массива содержит адрес диапазона, массив или пустую строку.

Обратите внимание, что нет специальной функции определения четвертого аргумента формулы SERIES (порядок построения). Этот аргумент определяется напрямую с помощью свойства PlotOrder объекта Series.

НА КОМПАКТ-ДИСКЕ

Полный код этого приложения слишком длинный, чтобы приводить его в книге. Но вы всегда можете ознакомиться с ним, воспользовавшись прилагаемым к книге компакт-диском.

В следующем примере отображается адрес массива значений первого ряда активной диаграммы.

```
Sub ShowValuesRange()
    Dim Ser As Series
    Dim x As Variant
    Set Ser = ActiveChart.SeriesCollection(1)
    x = VALUES_FROM_SERIES(Ser)
    If x(1) = "Range" Then
        MsgBox Range(x(2)).Address
    End If
End Sub
```

Переменная x имеет тип данных Variant, поскольку она будет содержать двухэлементный массив, возвращаемый функцией VALUES_FROM_SERIES. Первый элемент массива x представлен строкой и определяет тип данных. Если это объект Range, то в диалоговом окне отображается его адрес, представленный вторым элементом массива x.

На рис. 18.6 показан еще один пример. На диаграмме представлено три ряда данных. Кнопки на рабочем листе выполняют макросы, которые расширяют и сокращают каждый диапазон (ряд).

Ниже приведен листинг процедуры `ContractAllSeries`. В этой процедуре циклически просматривается коллекция `SeriesCollection` и вызываются функции `XVALUES_FROM_SERIES` и `VALUES_FROM_SERIES` для получения текущего диапазона. Далее с помощью метода `Resize` изменяется размер диапазона.

```
Sub ContractAllSeries()
    Dim s As Series
    Dim Result As Variant
    Dim DRange As Range
    For Each s In ActiveSheet.ChartObjects(1).Chart.SeriesCollection
        Result = XVALUES_FROM_SERIES(s)
        If Result(1) = "Range" Then
            Set DRange = Range(Result(2))
            If DRange.Rows.Count > 1 Then
                Set DRange = DRange.Resize(DRange.Rows.Count - 1)
                s.XValues = DRange
            End If
        End If
        Result = VALUES_FROM_SERIES(s)
        If Result(1) = "Range" Then
            Set DRange = Range(Result(2))
            If DRange.Rows.Count > 1 Then
                Set DRange = DRange.Resize(DRange.Rows.Count - 1)
                s.Values = DRange
            End If
        End If
    Next s
End Sub
```

Процедура `ExpandAllSeries` очень проста. С ее помощью каждый диапазон разбивается на отдельные ячейки.

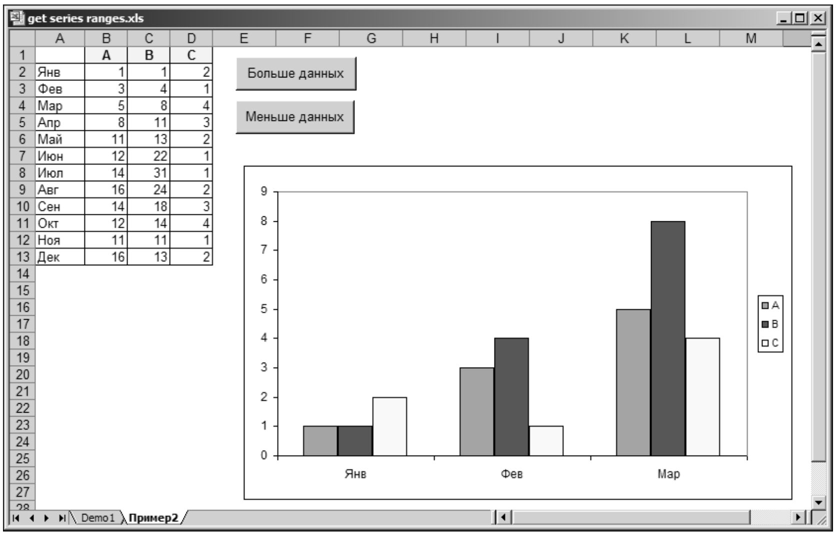


Рис. 18.6. Эта рабочая книга демонстрирует способы расширения и сокращения рядов данных

Отображение подписей для данных на диаграмме

Многие пользователи не удовлетворены несовершенством системы добавления подписей на диаграммы Excel. Например, рассмотрим график, который показан на рис. 18.7. Было бы неплохо отобразить подписи для каждой точки данных представленного диапазона. Но можно провести в поисках весь день, но так и не найти команды Excel, которая позволит разместить необходимые подписи в области диаграммы (такой команды просто не существует). Подписи к точкам данных ограничены только реальными значениями, если, конечно, не отредактировать каждую точку вручную и не ввести самостоятельно требуемый текст.

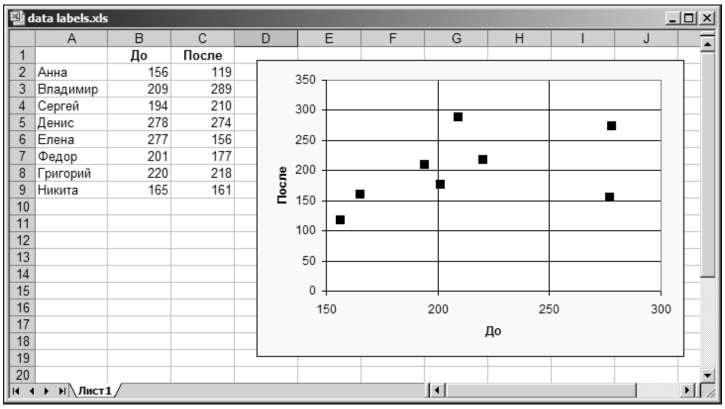


Рис. 18.7. График без подписей данных

Листинг 18.2 содержит простую процедуру, которая управляет первой диаграммой на активном листе. Она запрашивает у пользователя диапазон и просматривает коллекцию `Points` с целью заменить свойство `Text` каждого элемента коллекции на соответствующий фрагмент данных, который содержится в указанном диапазоне.

Листинг 18.2. Получение подписей для точек данных

```
Sub DataLabelsFromRange()
    Dim DLRange As Range
    Dim Cht As Chart
    Dim i As Integer, Pts As Integer

'   Определение диаграммы
    Set Cht = ActiveSheet.ChartObjects(1).Chart

'   Запрос диапазона
    On Error Resume Next
    Set DLRange = Application.InputBox _
      (prompt:="Укажите диапазон с подписями", Type:=8)
    If DLRange Is Nothing Then Exit Sub
    On Error GoTo 0

'   Добавление подписей
    Cht.SeriesCollection(1).ApplyDataLabels _
      Type:=xlDataLabelsShowValue, _
      AutoText:=True, _
      LegendKey:=False
```

```
'   Просмотр диапазона и назначение подписей
    Pts = Cht.SeriesCollection(1).Points.Count
    For i = 1 To Pts
        Cht.SeriesCollection(1). _
          Points(i).DataLabel.Text = DLRange(i)
    Next i
End Sub
```

Этот пример доступен на прилагаемом к книге компакт-диске.

На рис. 18.8 показана диаграмма после запуска процедуры `DataLabelsFromRange` и указания в качестве исходных данных диапазона `A2:A9`.

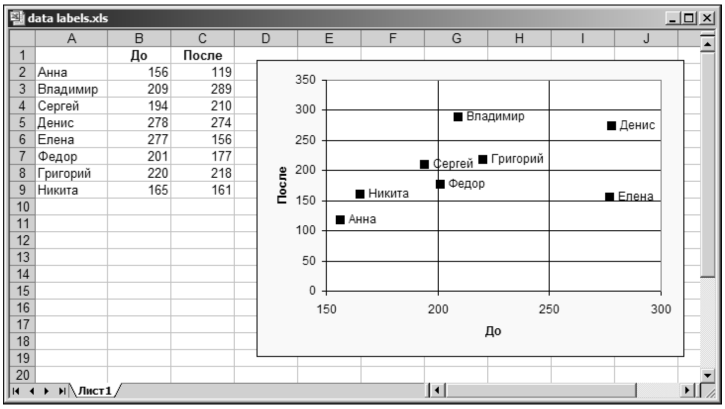


Рис. 18.8. Точечная диаграмма, которая содержит подписи для точек данных (достигается с помощью программирования на VBA)

Предыдущая процедура является достаточно грубой. Кроме того, с ее помощью нельзя выполнить глубокую проверку ошибочных состояний. Данная процедура работает только с первым объектом `Series`. Пакет Power Utility Pak содержит намного более сложный инструмент создания подписей для данных на диаграмме.

Отображение диаграммы в пользовательском диалоговом окне

В главе 15 рассмотрен способ отображения диаграммы в пользовательском диалоговом окне. Описанный метод заключается в сохранении диаграммы в виде файла формата GIF, после чего необходимо загрузить последний в элемент управления `Image`, размещаемый в диалоговом окне `UserForm`.

В примере, приведенном в этом разделе, используется именно этот подход. Но существует одна особенность: диаграмма создается “на лету” и строится на основе данных строки активной ячейки (рис. 18.9).

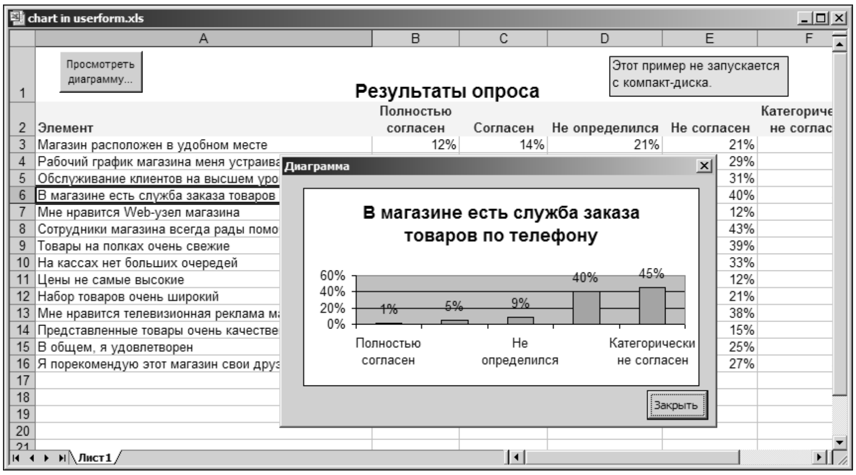


Рис. 18.9. Диаграмма в этом пользовательском диалоговом окне создается "на лету" на основе данных активной строки

Диалоговое окно `UserForm` этого примера довольно простое. Оно содержит элемент управления `Image` и элемент управления `CommandButton` (кнопка Закрыть). В рабочей книге, в которой находятся данные, вы найдете кнопку, щелчок на которой выполняет следующую процедуру.

```
Sub ShowChart()
    Dim UserRow As Long
    UserRow = ActiveCell.Row
    If UserRow < 2 Or IsEmpty(Cells(UserRow, 1)) Then
        MsgBox "Переместите курсор к ячейке с данными диапазона"
        Exit Sub
    End If
    CreateChart (UserRow)
    UserForm1.Show
End Sub
```

Так как диаграмма основана на данных строки активной ячейки, процедура отображает предупреждение, если курсор оказывается в неправильной строке данных. Если активная строка выбрана правильно, то процедура `ShowChart` вызывает процедуру `CreateChart` для создания необходимой диаграммы. После этого отображается пользовательское диалоговое окно.

Процедура `CreateChart`, представленная в листинге 18.3, принимает один аргумент, который представляет строку активной ячейки. Эта процедура получена на основе макроса, который записан при создании диаграммы и подкорректирован для приведения к более универсальному виду.

Листинг 18.3. Автоматическая генерация диаграммы

```
Sub CreateChart(r)
    Dim TempChart As Chart
    Dim CatTitles As Range
    Dim SrcRange As Range, SourceData As Range

    Application.ScreenUpdating = False
```

```
    Set CatTitles = ActiveSheet.Range("A2:F2")
    Set SrcRange = ActiveSheet.Range(Cells(r, 1), Cells(r, 6))
    Set SourceData = Union(CatTitles, SrcRange)

'   Добавление диаграммы
    Set TempChart = Charts.Add

'   Настройка диаграммы
    With TempChart
        .ChartType = xlColumnClustered
        .SetSourceData Source:=SourceData, PlotBy:=xlRows
        .HasLegend = False
        .ApplyDataLabels Type:=xlDataLabelsShowValue, LegendKey:=False
        .ChartTitle.Font.Size = 14
        .ChartTitle.Font.Bold = True
        .Axes(xlValue).MaximumScale = 0.6
        .Axes(xlCategory).TickLabels.Font.Size = 10
        .Axes(xlCategory).TickLabels.Orientation = xlHorizontal
        .Location Where:=xlLocationAsObject, Name:="Лист1"
    End With

'   Задание размера диаграммы
    With ActiveSheet.ChartObjects(1)
        .Width = 300
        .Height = 150
    End With
End Sub
```

После завершения работы процедуры `CreateChart` рабочий лист будет содержать объект `ChartObject` с диаграммой данных на основе строки активной ячейки. Но объект `ChartObject` останется невидимым, так как свойство `ScreenUpdating` объекта `Application` отключено.

Последний оператор в процедуре `ShowChart` загружает диалоговое окно `UserForm`. Ниже приведен листинг процедуры `UserForm_Initialize`. Эта процедура сохраняет диаграмму в виде файла GIF, удаляет объект `ChartObject` и загружает файл формата GIF в элемент управления `Image`.

```
Private Sub UserForm_Initialize()
    Dim CurrentChart As Chart
    Dim Fname As String

    Set CurrentChart = ActiveSheet.ChartObjects(1).Chart

'   Сохранение диаграммы в виде файла GIF
    Fname = ThisWorkbook.Path & Application.PathSeparator & "temp.gif"
    CurrentChart.Export FileName:=Fname, FilterName:="GIF"
    ActiveSheet.ChartObjects(1).Delete

'   Отображение диаграммы
    Image1.Picture = LoadPicture(Fname)
    Application.ScreenUpdating = True
    Kill ThisWorkbook.Path & Application.PathSeparator & "temp.gif"
End Sub
```

НА КОМПАКТ-ДИСКЕ

Данная рабочая книга доступна на прилагаемом компакт-диске.

События диаграмм

Программа Excel поддерживает несколько событий, связанных с диаграммами. Например, когда диаграмма активизируется, генерируется событие `Activate`. Событие `Calculate` возникает после того, как диаграмма получает новые или изменившиеся данные. Можно создать код VBA, который будет выполняться при возникновении определенного события.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Обратитесь к главе 19 для получения дополнительной информации о событиях.

В табл. 18.1 отображен список событий диаграммы, которые поддерживаются в Excel 97 и более поздних версиях программы.

Таблица 18.1. События, которые распознаются объектами диаграммы

Событие	Действие, которое приводит к возникновению события
`Activate`	Активизирован лист диаграммы или встроенная диаграмма
`BeforeDoubleClick`	На встроенной диаграмме выполнен двойной щелчок. Это событие происходит перед принятой по умолчанию реакцией на двойной щелчок
`BeforeRightClick`	На встроенной диаграмме выполнен щелчок правой кнопкой мыши. Это событие происходит перед принятой по умолчанию реакцией на щелчок правой кнопкой мыши
`Calculate`	На диаграмме отображаются новые или обновленные данные
`Deactivate`	Диаграмма деактивизируется
`DragOver`	Диапазон ячеек перетаскивается над диаграммой
`DragPlot`	Диапазон ячеек перетаскивается и отпускается на диаграмму
`MouseDown`	Кнопка мыши нажата, а указатель находится над диаграммой
`MouseMove`	Расположение указателя мыши изменяется, пока он находится над диаграммой
`MouseUp`	Кнопка мыши отпущена, пока указатель находится над диаграммой
`Resize`	Изменение размера диаграммы
`Select`	Выделение одного из элементов диаграммы
`SeriesChange`	Изменяется значение точки данных на диаграмме

Пример использования событий объекта Chart

Для того чтобы создать процедуру обработки события, которое происходит на листе диаграммы, код VBA должен быть сохранен в модуле кода объекта `Chart`. Чтобы активизировать этот модуль кода, дважды щелкните на элементе `Chart` в окне проектов. После этого в модуле кода выберите `Chart` из раскрывающегося списка Object в левой части окна и укажите событие в раскрывающемся списке Procedure, находящемся в правой части экрана (рис. 18.10).

НА ЗАМЕТКУ

Так как для встроенных диаграмм модуль кода не представлен, процедура, описанная в этом разделе, применяется только по отношению к листам диаграмм. Конечно, события можно обрабатывать и для встроенных диаграмм, но для этого необходимо выполнить кропотливую работу, связанную с созданием модуля класса. Такая процедура рассматривается далее в разделе “Поддержка событий для встроенных диаграмм”.

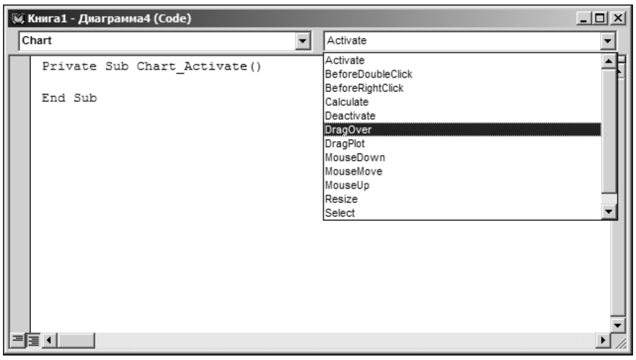


Рис. 18.10. Выбор события в модуле кода для объекта диаграммы

Пример, представленный ниже, отображает сообщение в следующих случаях: когда пользователь активизирует лист диаграммы; деактивизирует лист диаграммы; выделяет один из элементов диаграммы. Мною была создана рабочая книга с листом диаграммы, а также три процедуры обработки событий.

- `Chart_Activate` — выполняется, когда лист диаграммы активизируется.
- `Chart_Deactivate` — выполняется, когда лист диаграммы деактивизируется.
- `Chart_Select` — выполняется, когда выделен элемент листа диаграммы.

Ниже приведен код процедуры `Chart_Activate`.

```
Private Sub Chart_Activate()
    Dim msg As String
    msg = "Привет, " & Application.UserName & vbCrLf & vbCrLf
    msg = msg & "Вы просматриваете отчет по объемам продаж "
    msg = msg & "за 6 месяцев для 3 товаров" & vbCrLf & vbCrLf
    msg = msg & "Щелкните на элементе диаграммы для его идентификации"
    MsgBox msg, vbInformation, ActiveWorkbook.Name
End Sub
```

Эта процедура отображает сообщение при активизации диаграммы (рис. 18.11).

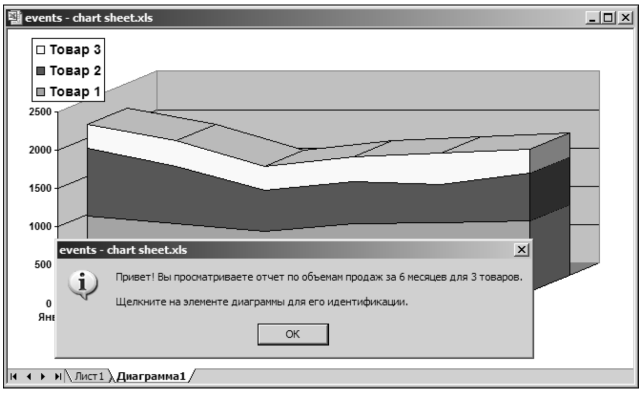


Рис. 18.11. Активизация диаграммы приводит к запуску процедуры `Chart_Activate` и отображению окна сообщения

Процедура Chart_Deactivate, которая представлена ниже, также отображает окно сообщения, однако лишь в том случае, когда диаграмма деактивизируется.

```
Private Sub Chart_Activate()
    Dim msg As String
    msg = "Привет! "
    msg = msg & "Вы просматриваете отчет по объемам продаж "
    msg = msg & "за 6 месяцев для 3 товаров" & vbCrLf & vbCrLf
    msg = msg & "Щелкните на элементе диаграммы для его идентификации"
    MsgBox msg, vbInformation, ActiveWorkbook.Name
End Sub
```

Процедура Chart_Select, приведенная ниже, выполняется каждый раз, когда пользователь выделяет один из элементов диаграммы.

```
Private Sub Chart_Select(ByVal ElementID As Long, _
  ByVal Arg1 As Long, ByVal Arg2 As Long)
    Dim Id As String
    Select Case ElementID
        Case xlChartArea: Id = "Область диаграммы"
        Case xlChartTitle: Id = "Заголовок диаграммы"
        Case xlPlotArea: Id = "Область построения"
        Case xlLegend: Id = "Легенда"
        Case xlFloor: Id = "Основание"
        Case xlWalls: Id = "Стенки"
        Case xlCorners: Id = "Углы"
        Case xlDataTable: Id = "Таблица данных"
        Case xlSeries: Id = "Последовательности"
        Case xlDataLabel: Id = "Подпись данных"
        Case xlTrendline: Id = "Тенденция"
        Case xlErrorBars: Id = "Погрешность"
        Case xlXErrorBars: Id = "Горизонтальная погрешность"
        Case xlYErrorBars: Id = "Вертикальная погрешность"
        Case xlLegendEntry: Id = "Запись легенды"
        Case xlLegendKey: Id = "Ключ легенды"
        Case xlAxis: Id = "Оси"
        Case xlMajorGridlines: Id = "Базовые линии сетки"
        Case xlMinorGridlines: Id = "Вспомогательные линии сетки"
        Case xlAxisTitle: Id = "Названия осей"
        Case xlShape: Id = "Фигура"
        Case xlNothing: Id = "Ничего"
        Case Else: Id = "Нечто другое"
    End Select
    MsgBox "Выделено: " & Id
End Sub
```

Эта процедура отображает сообщение, в котором находится описание выделенного элемента. При возникновении события Select аргумент ElementID содержит целое число, которое соответствует выделенному элементу. Аргументы Arg1 и Arg2 предоставляют дополнительную информацию о выделенном элементе (информацию о значениях этих аргументов можно получить в интерактивном справочном руководстве). Структура Select Case преобразует встроенные константы в описательные строки.

Это далеко не полный список элементов объекта Chart. Например, в нем отсутствуют элементы сводных диаграмм. Именно поэтому в листинге используется оператор Case Else.

Поддержка событий для встроенных диаграмм

Как отмечалось в предыдущем разделе, события объекта Chart автоматически реализованы для листов диаграмм, но отключены для диаграмм, встроенных в рабочий лист. Для того чтобы использовать события, связанные со встроенной диаграммой, необходимо выполнить следующие инструкции.

СОЗДАТЬ МОДУЛЬ КЛАССА

В VBE выберите проект в окне Project и выполните команду Insert⇨Class Module. Это приведет к добавлению пустого модуля класса в проект. Если возникнет желание, можно назначить модулю класса более описательное имя в окне Properties.

ОБЪЯВИТЕ ГЛОБАЛЬНЫЙ ОБЪЕКТ CHART

Следующим шагом является создание глобальной переменной, которая используется в качестве имени класса. Переменная должна иметь тип Chart и объявляться в модуле класса с ключевым словом WithEvents. Если опустить ключевое слово WithEvents, то объект не будет реагировать на события. Ниже приведен пример соответствующего объявления.

```
Public WithEvents myChartClass As Chart
```

СВЯЖИТЕ ОБЪЯВЛЕННЫЙ ОБЪЕКТ С ДИАГРАММОЙ

Перед тем, как будут запущены процедуры обработки событий, необходимо связать объявленный объект в модуле класса со встроенной диаграммой. Для этого следует объявить объект типа Class1 (в данном случае используется имя модуля класса). Это должна быть переменная уровня модуля, которая объявлена в обычном модуле VBA (а не в модуле класса).

```
Dim MyChart As New MyChartClass
```

Затем введите код, который фактически создаст экземпляр объекта. Воспользуйтесь представленным ниже оператором.

```
Set myChart.myChartClass = ActiveSheet.ChartObjects(1).Chart
```

После выполнения предыдущего оператора объект myChartClass в модуле класса будет указывать на первую встроенную диаграмму активного листа. Следовательно, процедуры обработки событий в модуле класса будут выполняться при возникновении соответствующих событий.

СОЗДАЙТЕ ПРОЦЕДУРЫ ОБРАБОТКИ СОБЫТИЙ ДЛЯ КЛАССА ДИАГРАММЫ

В этом разделе речь пойдет о том, как создавать процедуры обработки событий в модуле класса. Помните, что модуль класса должен содержать следующее объявление.

```
Public WithEvents myChartClass As Chart
```

После объявления с помощью ключевого слова WithEvents новый объект появится в раскрывающемся списке Object в модуле класса. Если вы выберете новый объект в поле Object, его действительные события будут перечислены в раскрывающемся списке Procedure в правой части окна (рис. 18.12).

Приведенный далее пример является простой процедурой обработки события, которая выполняется при активизации встроенной диаграммы. Эта процедура создает окно сообщения, которое содержит имя “родителя” объекта Chart (в его роли выступает объект ChartObject).

```
Private Sub clsChart_Activate()
    MsgBox clsChart.Parent.Name & " активизирована"
End Sub
```

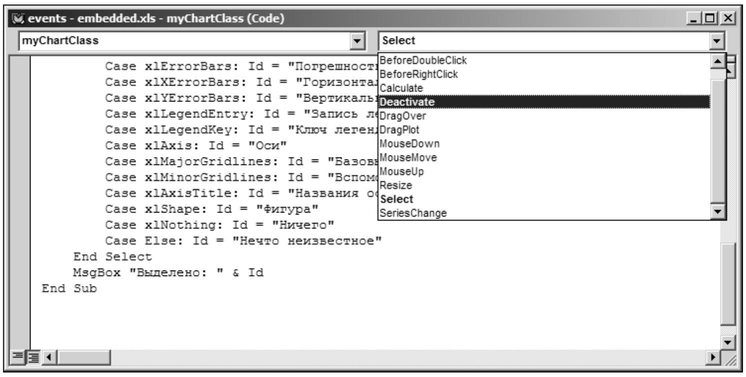


Рис. 18.12. Список Procedure *отображает действительные события для нового объекта* `Chart`

На прилагаемом к книге компакт-диске расположен файл рабочей книги, которая демонстрирует концепции, описанные в этом разделе.

Пример использования событий объекта Chart во встроенной диаграмме

Пример, приводимый в этом разделе, призван закрепить материал, приведенный выше. На рис. 18.13 отображена встроенная диаграмма, которая работает подобно карте изображения. На ее областях можно щелкать для получения разных результатов. Щелчок на одном из столбцов диаграммы приводит к активизации того рабочего листа, на котором содержатся подробные данные об этой области диаграммы.

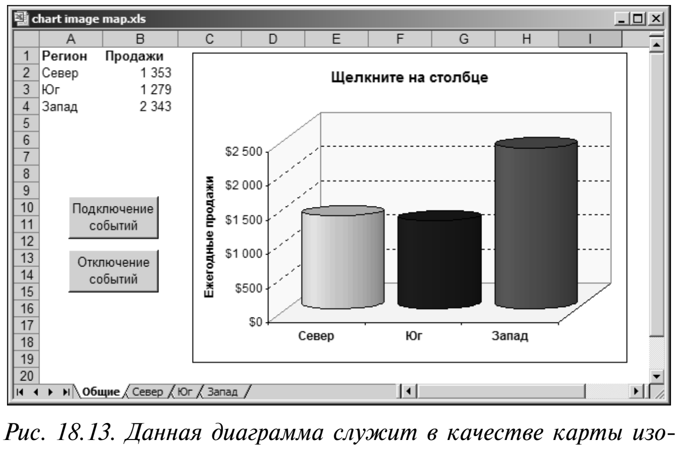


Рис. 18.13. Данная диаграмма служит в качестве карты изображения, на областях которой можно щелкать для отображения детальных сведений

Рабочая книга состоит из четырех листов. Лист Общие содержит встроенную диаграмму. Другие листы называются Север, Юг и Запад. Формулы в диапазоне B1:B4 суммируют данные в соответствующих листах, а итоговые данные отображаются на диаграмме. Щелчок на столбце диаграммы приводит к возникновению события. Процедура обработки этого события активизирует соответствующий лист, чтобы пользователь мог просмотреть подробные сведения выбранного диапазона.

В рабочей книге расположен модуль класса, который называется EmbChartClass, а также обычный модуль VBA — Module1. В демонстрационных целях рабочий лист Общие содержит две кнопки: одна выполняет процедуру EnableChartEvents, а вторая — процедуру DisableChartEvents (обе расположены в модуле Module1). Кроме того, на каждом рабочем листе находится кнопка, с помощью которой запускается процедура ReturntoMain, возвращающая пользователя на лист Общие.

Полный листинг модуля Module1 приведен ниже.

```
Dim SummaryChart As New EmbChartClass

Sub EnableChartEvents()
'   Вызывается кнопкой на рабочем листе
    Range("A1").Select
    Set SummaryChart.myChartClass = _
      Worksheets(1).ChartObjects(1).Chart
End Sub

Sub DisableChartEvents()
'   Вызывается кнопкой на рабочем листе
    Set SummaryChart.myChartClass = Nothing
    Range("A1").Select
End Sub

Sub ReturnToMain()
'   Вызывается кнопкой на рабочем листе
    Sheets("Общие").Activate
End Sub
```

Первый оператор объявляет новый объект SummaryChart типа EmbChartClass (это имя модуля класса). Когда пользователь щелкает на кнопке Подключение событий, встроенная диаграмма присваивается объекту SummaryChart, который, в свою очередь, поддерживает обработку событий диаграммы. В листинге 18.4 представлен код модуля класса EmbChartClass.

Щелчок на диаграмме приводит к генерации события MouseDown, которое вызывает выполнение процедуры myChartClass_MouseDown. Эта процедура использует метод GetChartElement для получения информации об элементе, на котором щелкнул пользователь. Метод GetChartElement возвращает информацию об элементе, которому принадлежит точка с координатами X и Y под указателем мыши во время щелчка (информация о координатах отображается в виде аргументов процедуры myChartClass_MouseDown).

Листинг 18.4. Реакция на щелчок на столбце

```
Public WithEvents myChartClass As Chart

Private Sub myChartClass_MouseDown(ByVal Button As Long, _
  ByVal Shift As Long, ByVal X As Long, ByVal Y As Long)

    Dim IDnum As Long
    Dim a As Long, b As Long

'   Следующий оператор возвращает значения
'   IDNum, a и b
```

```
    myChartClass.GetChartElement X, Y, IDnum, a, b

'   Щелчок на последовательности?
    If IDnum = xlSeries Then
        Select Case b
            Case 1
                Sheets("Север").Activate
            Case 2
                Sheets("Юг").Activate
            Case 3
                Sheets("Запад").Activate
        End Select
    End If
    Range("A1").Select
End Sub
```

Эта рабочая книга доступна на прилагаемом к книге компакт-диске.

Хитрости создания диаграмм

Далее мы предложим информацию, касающуюся тонкостей создания диаграмм. Некоторые из описанных методов могут эффективно применяться в конечных приложениях. Их изучение предоставит дополнительную информацию об объектной модели диаграмм.

Печать встроенных диаграмм на всю страницу

При выделении встроенной диаграммы можно распечатать диаграмму с помощью команды Файл⇨Печать. Встроенная диаграмма будет распечатана на всю страницу (как будто она находится на листе диаграммы), но при этом останется встроенной. Следующий макрос печатает все встроенные диаграммы на активном листе, причем каждая диаграмма печатается на всю страницу.

```
Sub PrintEmbeddedCharts()
    For Each chtObj In ActiveSheet.ChartObjects
        chtObj.Chart.Print
    Next chtObj
End Sub
```

Создание “мертвой” диаграммы

Как правило, диаграмма Excel использует данные, которые хранятся в диапазоне. Изменение данных в диапазоне приводит к автоматическому обновлению содержимого диаграммы. В некоторых случаях возникает необходимость “отсоединить” диаграмму от ее диапазона данных и создать “мертвую” диаграмму (чтобы она никогда не изменялась). Например, если данные диаграммы создаются рядом сценариев “что-если”, то может возникнуть необходимость в сохранении базовой диаграммы, которая сравнивается с результатами выполнения других сценариев.

Для создания такой диаграммы существует два метода.

- *Вставить в виде изображения*. Активизируйте диаграмму и выберите Правка⇨Копировать. После этого нажмите клавишу <Shift> и выполните команду Правка⇨Вставить рисунок (команда Вставить рисунок доступна только в том случае, если при открытии меню Правка удерживать нажатой клавишу <Shift>). В результате будет получено изображение скопированной диаграммы.

♦ *Преобразовать ссылки на диапазоны в массивы значений.* Щелкните на последовательности в диаграмме, а затем — в строке формул. Нажмите клавишу <F9>, чтобы преобразовать диапазон в массив. Повторите эту операцию для остальных последовательностей диаграммы.

Файл `xl8galry.xls` отображает последний метод. Данный файл является специальной рабочей книгой, которая используется Excel для хранения пользовательских форматов диаграмм. Если открыть эту рабочую книгу, то можно найти 20 листов диаграмм. Каждый лист диаграммы содержит “макеты” диаграмм, в которых вместо диапазонов в качестве источников используются массивы.

Приведенная ниже процедура создает изображение активной диаграммы поверх самой диаграммы (исходная диаграмма не удаляется). Она выполняется как для внедренных диаграмм, так и для диаграмм на отдельных листах.

```
Sub ConvertChartToPicture()
    Dim TheChart As Chart
    If ActiveChart Is Nothing Then
        MsgBox "Activate a chart.", vbInformation
        Exit Sub
    End If
    Set TheChart = ActiveChart
    Application.ScreenUpdating = False
    TheChart.CopyPicture Appearance:=xlPrinter, _
      Size:=xlScreen, Format:=xlPicture
    If TypeName(ActiveSheet) = "Chart" Then
        ActiveChart.Paste
    Else
        ActiveSheet.Paste
        Selection.Left = TheChart.Parent.Left
        Selection.Top = TheChart.Parent.Top
    End If
    Application.ScreenUpdating = True
End Sub
```

Соответствующая рабочая книга расположена на прилагаемом к книге компакт-диске.

Следующая процедура создает диаграмму (рис. 18.14), управляя несколькими массивами. Рабочие данные не сохраняются на рабочем листе. Более того, формула `SERIES` (`РЯД`) оперирует массивами, а не ссылками на диапазоны.

```
Sub CreateADeadChart()
    Charts.Add
    ActiveChart.Location _
      Where:=xlLocationAsObject, Name:="Sheet1"
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(1).XValues = Array("Jan", "Feb", "Mar")
        .SeriesCollection(1).Values = Array(125, 165, 189)
        .SeriesCollection(1).Name = "Sales"
        .ChartType = xlColumnClustered
        .HasLegend = False
        .Deselect
    End With
End Sub
```

Соответствующая рабочая книга расположена на прилагаемом к книге компакт-диске.

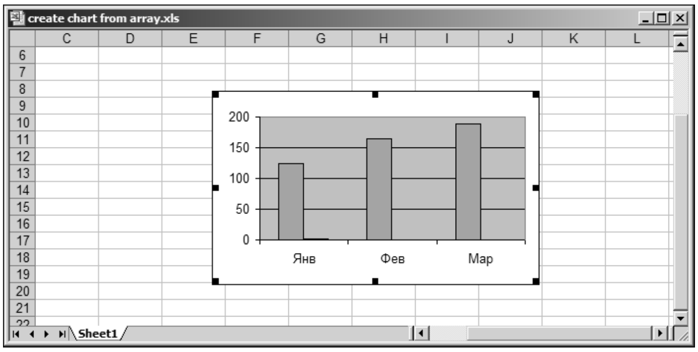


Рис. 18.14. Эта диаграмма построена на основе данных массивов

Поскольку формула SERIES (РЯД) имеет ограничение на длину, то подобный подход можно использовать только для небольших наборов данных.

Отображение подсказки

В диаграммах очень удобно использовать экранные подсказки. Экранные подсказки — это небольшие сообщения, которые отображаются на экране при наведении указателя мыши на элемент диаграммы. Обычно в экранной подсказке представлено имя элемента (для рядов данных) и значение точки данных. Объект Chart не позволяет управлять экранными подсказками для диаграммы.

СОВЕТ Для отображения и скрытия экранных подсказок выполните команду Сервис⇨Параметры. Перейдите на вкладку Диаграмма и снимите или установите два флажка в разделе Всплывающие подсказки отображают.

В этом разделе описан способ управления экранными подсказками. На рис. 18.15 показана экранная подсказка, отображенная с помощью события MouseOver. При наведении указателя мыши на столбец в левом верхнем углу диаграммы появится текстовое окно с информацией о выбранной точке данных.

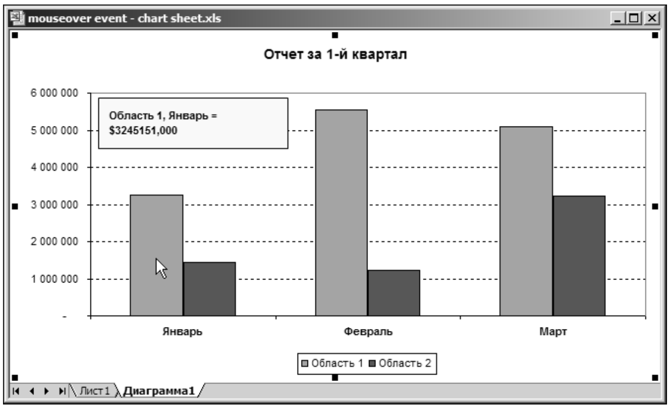


Рис. 18.15. Текстовое окно содержит сведения о выбранной точке данных

Процедура обработки событий расположена в модуле листа `Chart`.

```
Private Sub Chart_MouseMove(ByVal Button As Long, ByVal Shift As Long, _
  ByVal X As Long, ByVal Y As Long)
    Dim ElementId As Long
    Dim arg1 As Long, arg2 As Long
    Dim NewText As String
    On Error Resume Next
    ActiveChart.GetChartElement X, Y, ElementId, arg1, arg2
    If ElementId = xlSeries Then
        NewText = Sheets("Лист1").Range("Comments").Offset(arg2, arg1)
    Else
        NewText = ""
        ActiveChart.Shapes(1).Visible = False
    End If
    If NewText <> ActiveChart.Shapes(1).TextFrame.Characters.Text Then
        ActiveChart.Shapes(1).TextFrame.Characters.Text = NewText
        ActiveChart.Shapes(1).Visible = True
    End If
End Sub
```

Эта процедура отслеживает движения указателя мыши на листе диаграммы. Координаты указателя содержатся в переменных `X` и `Y`, подставляемых в процедуру. Аргументы `Button` и `Shift` в процедуре не используются.

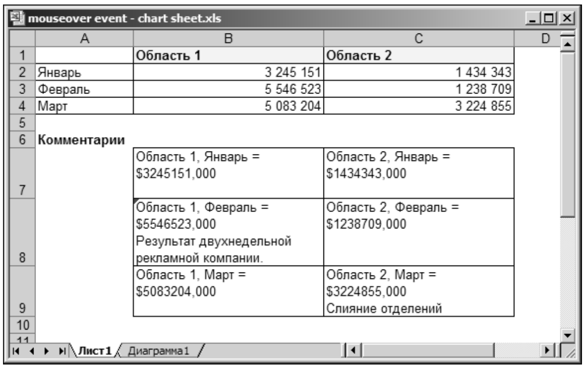
Ключевой элемент процедуры — это метод `GetChartElement`. Если переменная `ElementId` равна `xlSeries`, то указатель наведен на ряд данных. В результате переменной `NewText` назначается текст определенной ячейки с описательной информацией (рис. 18.16). Если указатель мыши находится за пределами ряда данных, то текст скрывается.

НА КОМПАКТ-ДИСКЕ

Эта рабочая книга доступна на прилагаемом к книге компакт-диске.

Анимированные диаграммы

Excel отображает простую анимацию (многие пользователи не знают об этой возможности). Например, можно анимировать фигуры и графики. Рассмотрим график, показанный на рис. 18.17.


mouseover event - chart sheet.xls

	A	B	C	D
1		**Область 1**	**Область 2**	
2	Январь	3 245 151	1 434 343	
3	Февраль	5 546 523	1 238 709	
4	Март	5 083 204	3 224 855	
5				
6	**Комментарии**			
7		Область 1, Январь = $3245151,000	Область 2, Январь = $1434343,000	
8		Область 1, Февраль = $5546523,000 Результат двухнедельной рекламной компании.	Область 2, Февраль = $1238709,000	
9		Область 1, Март = $5083204,000	Область 2, Март = $3224855,000 Слияние отделений	
10				

Лист1 / Диаграмма1

Рис. 18.16. Диапазон B7:C9 содержит описательные сведения, отображаемые в экранной подсказке

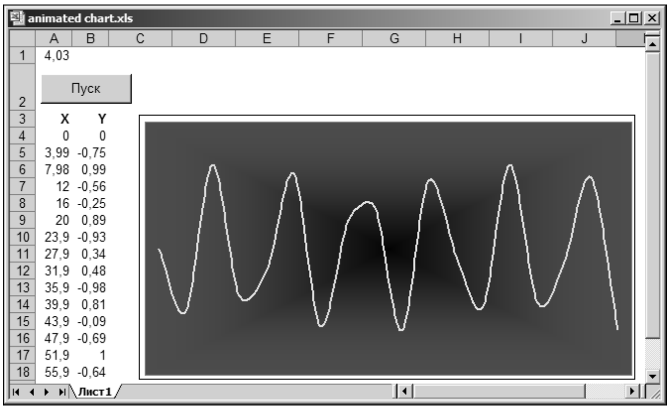


Рис. 18.17. С помощью простой процедуры VBA на основе этого графика можно получить интересную анимационную картинку

Значения аргумента X (столбец A) зависят от значения, которое хранится в ячейке A1. Значение в каждой строке равно сумме значения в предыдущей строке и значения в ячейке A1. Столбец B содержит формулы, которые подсчитывают значение SIN для соответствующего значения в столбце A. Представленная далее простая процедура создает интересную анимационную последовательность. Она изменяет значение в ячейке A1, что приводит к изменению диапазона значений X и Y.

Прилагаемый к книге компакт-диск содержит рабочую книгу, демонстрирующую анимированную диаграмму, а также несколько других примеров анимации.

Создание диаграммы с графиком гипоциклоиды

Даже если тригонометрия в школе вызывала у вас трудности, пример, приведенный в этом разделе, вам точно понравится (он основан на тригонометрических функциях). Рабочая книга, показанная на рис. 18.18, содержит бесконечное количество гипоциклоидных кривых. *Гипоциклоида* — это траектория точки, находящейся на окружности, которая движется внутри другой окружности. Подобная техника использовалась в популярной детской игрушке “Спирограф”.

Данная рабочая книга доступна на прилагаемом к книге компакт-диске.

В приведенном примере используется простой график. Данные для аргументов X и Y генерируются на основе формул, которые хранятся в столбцах A и B. Элементы управления Scrollbar в верхней части листа позволяют менять значения трех параметров, которые определяют внешний вид диаграммы. Эти элементы управления связаны с ячейками B1, B2 и B3. Они вставлены с панели инструментов Формы и не являются элементами управления ActiveX. Кроме того, диаграмма имеет кнопку Произвольно, которая генерирует случайные значения параметров.

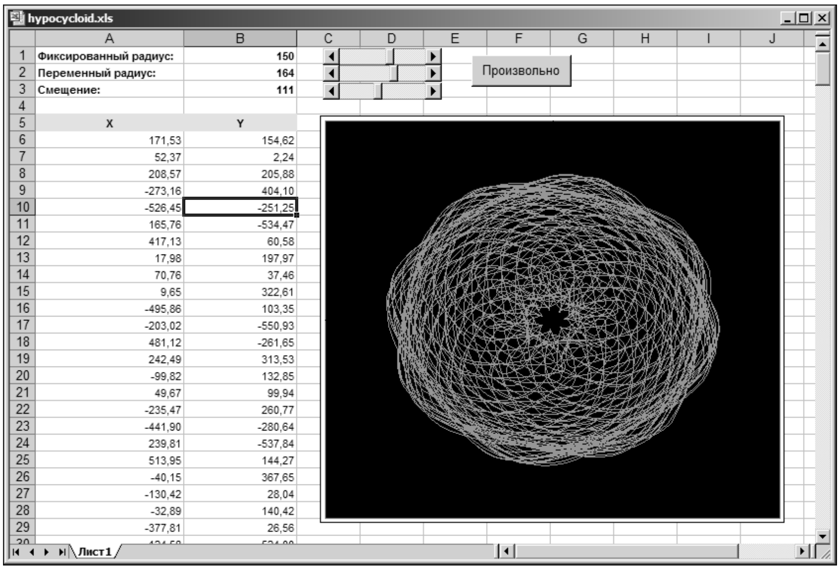


Рис. 18.18. Рабочая книга, генерирующая бесконечное количество гипоциклоидных кривых

Рабочая книга содержит только один макрос (показанный ниже). Он выполняется при щелчке на кнопке Произвольно. Этот макрос генерирует три случайных числа в диапазоне от 1 до 250 и вставляет эти числа в рабочий лист.

```
Sub Random_Click()
    Randomize
    Range("B1") = Int(Rnd * 250)
    Range("B2") = Int(Rnd * 250)
    Range("B3") = Int(Rnd * 250)
End Sub
```

Создание диаграммы часов

На рис. 18.19 показан график, который видоизменен так, чтобы имитировать циферблат часов. Такая диаграмма не только *выглядит*, как часы, она и работает, как часы. Сложно придумать область применения такой диаграммы, но ее создание — хорошее практическое упражнение.

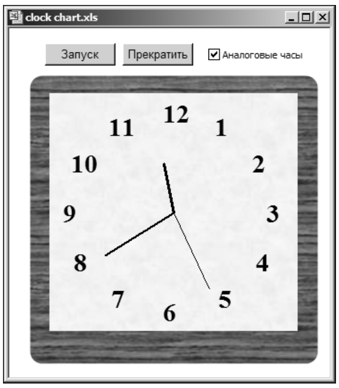


Рис. 18.19. Работающие часы в виде диаграммы

Данная рабочая книга доступна на прилагаемом к книге компакт-диске.

Кроме диаграммы часов, рабочая книга содержит текстовое поле, которое отображает время в виде строки, как показано на рис. 18.20. Обычно это поле скрыто, но его можно отобразить, сбросив флажок Аналоговые часы.

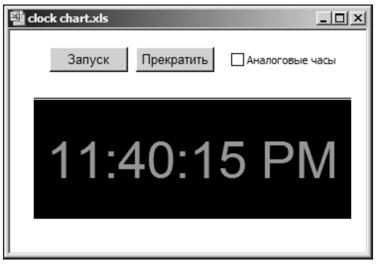


Рис. 18.20. Отобразить на листе цифровые часы намного проще, но результат будет не настолько эффектным, как в обычных

При рассмотрении рабочей книги, которая содержится на компакт-диске, обратите внимание на несколько факторов.

- Объект `ChartObject` называется `ClockChart` и использует диапазон `DigitalClock` для цифрового отображения времени.
- Две кнопки на листе добавлены с панели инструментов Формы. Каждой кнопке назначена процедура (`StartClock` и `StopClock`).
- Элемент управления `CheckBox` (Флажок) (с именем `cbClockType`) на листе взят с панели инструментов Форма, а не с панели инструментов Элементы управления. Щелчок на этом объекте приводит к выполнению процедуры `cbClockType_Click`, которая изменяет значение свойства `Visible` объекта `ClockChart`. Когда данный объект становится невидимым, пользователю открывается цифровой индикатор времени.
- В качестве диаграммы используется график с четырьмя объектами `Series`. Эти последовательности представляют часовую стрелку, минутную стрелку, секундную стрелку, а также 12 цифр.
- Процедура `UpdateClock` выполняется при щелчке на кнопке Запуск. Данная процедура определяет видимость циферблата и выполняет необходимое обновление экрана.
- Процедура `UpdateClock` использует метод `OnTime` объекта `Application`. Этот метод позволяет выполнять процедуру в определенное время. Перед завершением процедуры `UpdateClock` устанавливается новое событие `OnTime`, которое произойдет через одну секунду. Другими словами, процедура `UpdateClock` вызывается каждую секунду.
- Процедура `UpdateClock` использует основные тригонометрические функции для определения углов отображения стрелок часов.
- В отличие от многих других диаграмм, эта диаграмма не использует диапазон ячеек рабочего листа для хранения собственных данных. Вместо этого необходимые данные рассчитываются с помощью кода VBA и передаются непосредственно в свойства `Values` и `XValues` объектов `Series` диаграммы.

Советы по использованию диаграмм, не требующие написания макросов

В этом разделе мы поговорим о том, как можно управлять диаграммами без использования макросов. Вы будете удивлены тем богатством средств, которые предоставляет Excel для управления диаграммами.

Управление последовательностями методом скрытия данных

На рис. 18.21 показана диаграмма, которая отображает данные для каждого из 365 дней в году. Итак, вам необходимо отобразить лишь данные для февраля. Вы можете, конечно, переопределить диапазон данных диаграммы. Существует и другой путь — воспользоваться командой Excel Автофильтр.

По умолчанию диаграмма не отображает данные, которые скрыты на рабочем листе. Команда Excel Автофильтр скрывает строки, не соответствующие определенному критерию. Предлагаемое ею решение достаточно очевидно и просто. Выберите Данные⇨Фильтр⇨Автофильтр, чтобы перейти в режим автофильтра. Каждый заголовок строки в фильтруемом списке содержит стрелку раскрывающегося списка. Щелкните на опции Условие. После этого введите критерии фильтрации, которые определяют даты, отображаемые на диаграмме. Установки, показанные на рис. 18.22, скрывают все строки, кроме тех, которые содержат данные февраля.

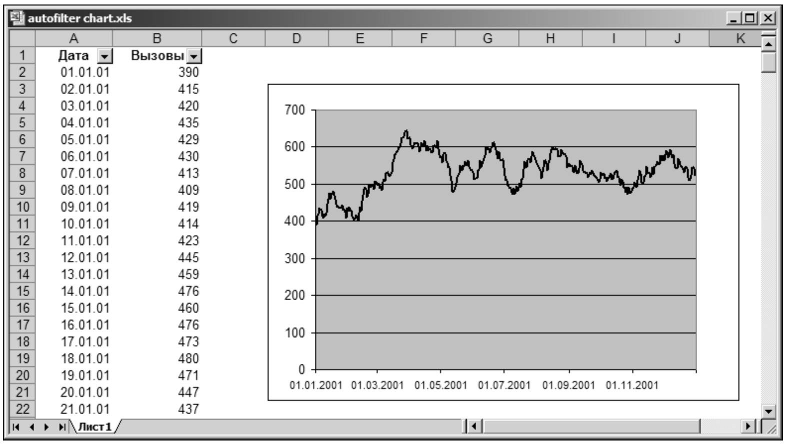


Рис. 18.11. Воспользуйтесь средством Автофильтр *для отображения только подмножества данных*

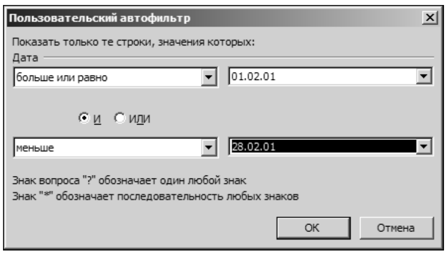


Рис. 18.22. В диалоговом окне Пользовательский автофильтр *указываются условия фильтрации данных листа*

Эта рабочая книга содержится на прилагаемом к книге компакт-диске.

Полученная диаграмма показана на рис. 18.23.

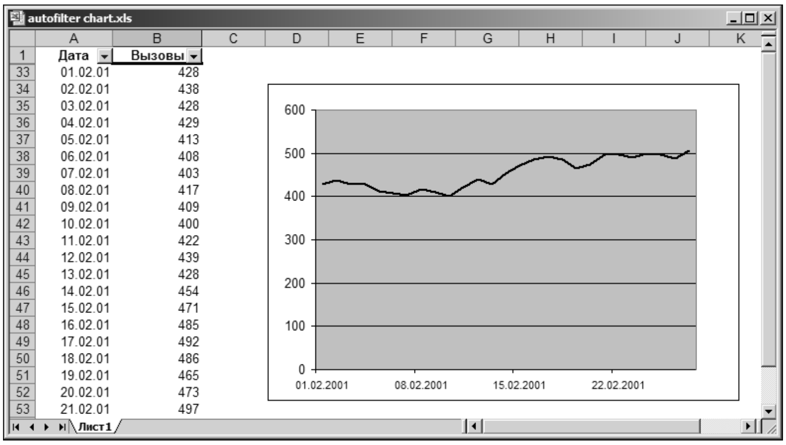


Рис. 18.23. На диаграмме представлены только данные отображаемых на экране ячеек

Если эта методика не работает, то необходимо изменить параметры диаграммы. Активизируйте диаграмму, после этого выберите команду Сервис⇨Параметры. В диалоговом окне Параметры щелкните на вкладке Диаграмма и установите флажок Отображать только видимые ячейки. Чтобы обеспечить отображение диаграммы даже при скрытии ячеек с данными, необходимо установить привязку диаграммы в значение не перемещать и не изменять размеры. Воспользуйтесь командой Формат⇨Выделенная область диаграммы для изменения этого параметра.

Хранение нескольких диаграмм на одном листе диаграммы

Пользователи Excel, которые задавались рассматриваемым в данном разделе вопросом, согласятся, что лист диаграммы содержит только одну диаграмму. Почти всегда это высказывание справедливо. Но хранение нескольких диаграмм на листе диаграммы возможно. Программа Excel предоставляет для этого определенную возможность. Если активизировать встроенную диаграмму и выбрать команду Диаграмма⇨Размещение, то Excel отобразит диалоговое окно Размещение диаграммы. Если выбрать переключатель Отдельном (и указать существующий лист диаграммы), то новая диаграмма будет размещена поверх той, которая уже расположена на листе диаграммы.

Зачастую возникает необходимость в размещении встроенной диаграммы на *пустом* листе диаграммы. Для того чтобы создать пустой лист диаграммы, выберите одну пустую ячейку и нажмите клавишу <F11>.

Одним из преимуществ хранения нескольких диаграмм на одном листе является то, что в таком случае можно использовать команду Вид⇨По размеру окна для автоматического масштабирования диаграмм в соответствии с размерами окна. На рис. 18.24 показан пример листа диаграммы, который содержит шесть встроенных диаграмм.

Данная рабочая книга находится на прилагаемом к книге компакт-диске.

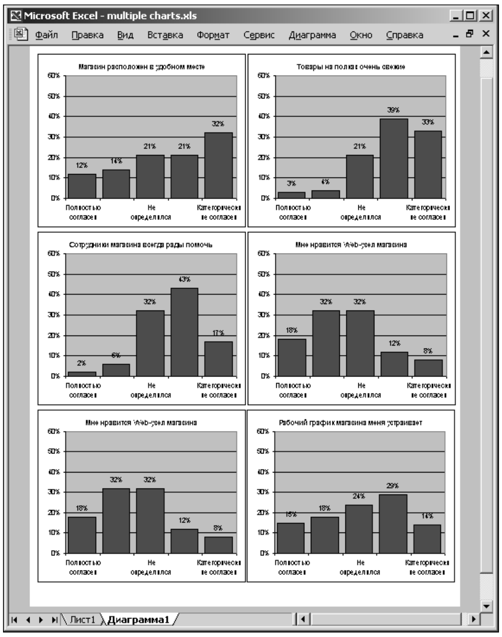


Рис. 18.24. Этот лист диаграммы содержит шесть встроенных диаграмм

Создание саморасширяющейся диаграммы

При создании диаграмм пользователи чаще всего задаются вопросом: как создать диаграмму, которая будет автоматически расширяться при добавлении в исходный диапазон новых данных?

Чтобы понять, о чем идет речь, взгляните на рис. 18.25. На нем показан рабочий лист с ежедневно обновляемыми сведениями об объемах продаж. На диаграмме графически представлены все данные рабочего листа. При введении новых данных диаграмма должна расширяться, чтобы вместить обновленную информацию. С другой стороны, при удалении данных диаграмма должна сужаться, чтобы исключить возможность образования пустых мест.

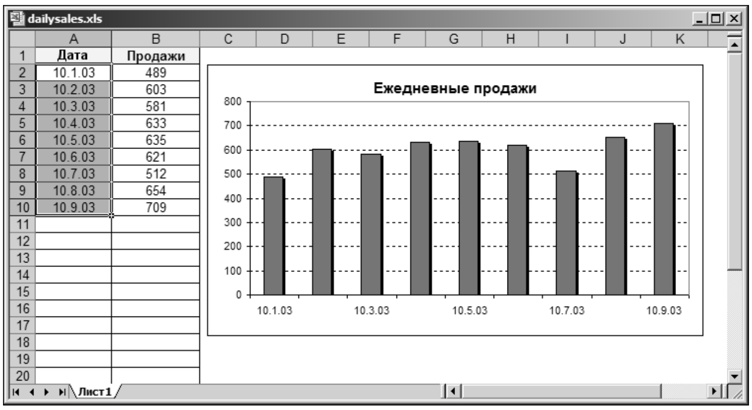


Рис. 18.25. В случае саморасширяющейся диаграммы обновление данных выполняется автоматически

При использовании Excel 2003 подобные ухищрения ни к чему. С помощью нового средства создания списков (команда Данные⇨Список⇨Создать список) вы сможете создать автоматически обновляемую диаграмму без выполнения дополнительных действий. Для расширения и сужения диаграммы достаточно изменить данные в назначенном списке.

Дело первостепенной важности — это создание диапазона, который способен вместить любые обновленные данные. В результате возникает еще одна проблема — на диаграмме отображаются "пустые" значения, которые только мешают правильному анализу данных. В большинстве случаев отображение на диаграмме "пустых" значений просто недопустимо.

Поскольку методика создания самообновляющейся диаграммы не так проста, как кажется на первый взгляд, мы поэтапно рассмотрим процесс ее реализации.

Для примера возьмем простой рабочий лист, в котором в столбце A расположены даты, а в столбце B указаны объемы продаж на указанную дату. Также предположим, что ежедневно рабочий лист пополняется новыми сведениями об объеме продаж, и эта информация должна незамедлительно отображаться на диаграмме.

СОЗДАНИЕ ДИАГРАММЫ

Для начала создадим диаграмму стандартного типа, на которой отображаются исходные данные. На рис. 18.25 показан рабочий лист, иллюстрирующий эту операцию. Диаграмма строится с помощью формулы РЯД следующего вида.

```
=РЯД(Лист1!$B$1;dailysales.xls!Дата;dailysales.xls!Продажи;1)
```

В формуле учтены следующие положения.

- Название ряда данных приведено в ячейке B1.
- Подписи категорий приведены в диапазоне A2:A11.
- Значения расположены в диапазоне B2:B11.

Как вы видите, это самая обычная диаграмма. Если добавить новую дату или значение объема продаж, то диаграмма не будет дополнена новыми элементами.

СОЗДАНИЕ ИМЕНОВАННОЙ ФОРМУЛЫ

На этом этапе мы создадим две именованные формулы. Они будут использоваться в качестве аргументов функции РЯД. Если вы еще не знакомы с синтаксисом функции РЯД, то будете иметь возможность ознакомиться с ним несколько позже. В данный момент выполните следующие действия.

1. Выберите команду Вставка⇨Имя⇨Присвоить. На экране появится диалоговое окно Присвоение имени.
2. В поле Имя введите Дата. В поле Формула введите следующую формулу.

   ```
   =СМЕЩ(Лист1!$A2;0;0;СЧЕТЗ(Лист1!$A:$A)-1;1)
   ```

3. Щелкните на кнопке Добавить для создания формулы с именем Дата.

   Обратите внимание, что функция СМЕЩ ссылается на первую подпись категорий (ячейка A2) и использует функцию СЧЕТЗ для определения количества подписей в столбце. Поскольку в столбце A в первой строке содержатся заголовки, то из общего количества вычитается значение 1.
4. В поле Имя введите Продажи, а в поле Формула — следующую формулу.

   ```
   =СМЕЩ(Лис-т1!$B$2;0;0;СЧЕТЗ(Лист1!$B:$B)-1;1)
   ```

В данном случае функция СМЕЩ обращается к первому значению данных (ячейка B2). Опять-таки, функция СЧЕТЗ определяет количество точек данных, за исключением заголовка в ячейке B1.

5. Щелкните на кнопке Добавить для добавления второй именованной формулы.
6. Щелкните на кнопке Закрыть, чтобы закрыть диалоговое окно присвоения имен.

Теперь ваша рабочая книга содержит две именованные формулы — Дата и Продажи.

ИЗМЕНЕНИЯ РЯДА ДАННЫХ

В заключение мы изменим диаграмму так, чтобы в ней использовались именованные формулы вместо строго заданных ссылок.

1. Активизируйте диаграмму и выполните команду Диаграмма⇨Исходные данные. На экране появится диалоговое окно Исходные данные.
2. В поле Значения введите Лист1!Продажи.
3. В поле Подписи оси X введите Лист1!Дата.
4. Убедитесь, что диалоговое окно выглядит так, как показано на рис. 18.26, и щелкните на кнопке OK.

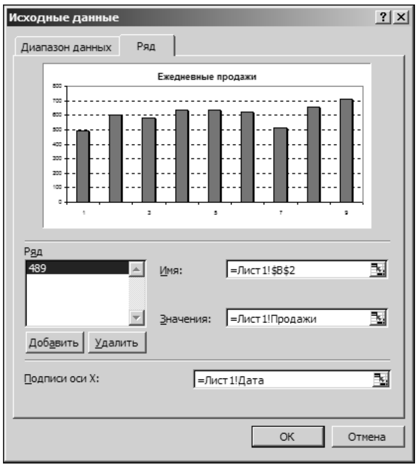


Рис. 18.26. Определив именованные формулы в качестве исходных данных, вы создадите диаграмму на основе изменяемого диапазона данных

В пп. 2–3 обратите внимание на то, что имя формулы начинается названием рабочего листа с данными. Поскольку именованная формула задается на уровне рабочей книги (в отличие от уровня рабочего листа), то вы должны были использовать имя рабочей книги. Но Excel в данном случае делает для вас исключение, корректно распознавая именованную формулу. Если же вы опять откроете диалоговое окно присвоения имени, то увидите, что программа автоматически заменила имя рабочего листа именем рабочей книги.

```
=dailysales.xls!Продажи
```

При создании именованной формулы вам необходимо начинать ее с названия рабочего листа или рабочей книги, к которым она относится. (Всегда проще использовать имя рабочего листа.) Помните, если имя рабочего листа или рабочей книги содержит пробелы, его следует заключить в одинарные кавычки.

```
='daily sales.xls'!Продажи
```

Детально об именах в Excel рассказано в следующей врезке.

Управление именами в Excel

В Excel поддерживается два типа имен: уровня рабочего листа и уровня рабочей книги. На уровне рабочей книги определяются имена, которые будут использоваться во всей книге. Обычно при назначении имени ячейке или диапазону вы создаете имя, доступное в рабочем листе.

Но вы можете создавать также имена только для отдельных рабочих листов. Подобное имя начинается с названия рабочего листа (например, имя `Лист1!Данные` определено на уровне рабочего листа). После создания допускается использовать имя без указания рабочего листа.

```
=Данные*4
```

Если же вы хотите ввести эту формулу на другом рабочем листе, то указывать в имени название исходного рабочего листа обязательно.

```
=Лист1!Данные*4
```

Имена уровня рабочего листа весьма эффективно использовать в рабочей книге с большим количеством одинаковых названий. Например, вы можете назначить одинаковые имена разным диапазонам на разных рабочих листах. Но при этом в каждое имя обязательно включите название рабочего листа, чтобы не вызвать конфликт данных.

Именованные формулы этой главы, в основном, задаются на уровне рабочей книги, поэтому название рабочего листа в них не указывается. Но при введении имени в поле Значения диалогового окна Исходные данные указывать имя рабочего листа обязательно.

СОВЕТ

Вместо редактирования данных с помощью диалогового окна Исходные данные, вы можете прибегнуть к непосредственному изменению формулы `РЯД`.

ТЕСТИРОВАНИЕ САМОРАСШИРЯЮЩЕЙСЯ ДИАГРАММЫ

Для проверки полученного результата введите дополнительные данные в столбцы `A` и `B` или удалите из них несколько значений. Если описанные выше инструкции выполнены вами правильно, то диаграмма будет обновлена автоматически. Если на экране отображается сообщение об ошибке, то проверьте правильность проведенных изменений. Этот метод достаточно прост, чтобы не сработать.

КАК ЭТО РАБОТАЕТ?

Многие пользователи применяют описанную выше методику, не понимая основ, на которых она базируется. Но в этом нет ничего сложного. Вам не обязательно вникать во все тонкости построения формул. В результате простых манипуляций вы сможете приспособить данную методику к любому типу диаграмм. Но зная, как создавались именованные формулы, вы сделаете это быстрее и проще. Кроме того, вам станут доступны динамические диаграммы более высокого уровня.

ОБ ИМЕНОВАННЫХ ФОРМУЛАХ В ДИАГРАММАХ

Описанная выше диаграмма использует технологию *именованных формул* Excel. Вы, возможно, уже знакомы с концепцией именованных ячеек и диапазонов, однако вы вряд ли догадываетесь, что последние — это частный случай именованных формул.

В диалоговом окне Присвоение имени в поле Имя вводится имя формулы, а в поле Формула — сама формула. Как несложно заметить, значение в поле формула начинается со знака равенства (а это явный признак формулы).

В отличие от обычной формулы именованная, формула не вводится в ячейку. Она сохраняется в памяти Excel. Результат выполнения именованной формулы отображается при запросе в рабочей книге самой формулы.

После определения именованные формулы вычисляются каждый раз при пересчете рабочего листа. Но эти формулы не сохраняются в ячейках, поэтому в явном виде результат их выполнения не виден. В нашем примере пересчет именованных формул вызывает изменение вида диаграммы, поскольку они участвуют в формуле `РЯД`.

Функция СМЕЩ

При создании обновляющейся диаграммы активно использовалась функция `СМЕЩ`. Она возвращает диапазон, смещенный относительно указанной ячейки. Аргументы функции указывают расстояние смещения относительно исходной ячейки и размер диапазона (количество столбцов и строк).

В функции `СМЕЩ` используется пять аргументов.

- `Ссылка` — ссылка, от которой вычисляется смещение. Аргумент `Ссылка` должен быть ссылкой на ячейку или на диапазон смежных ячеек; в противном случае функция `СМЕЩ` возвращает значение ошибки `#ЗНАЧ!`.
- `Смещ_по_строкам` — количество строк, которые нужно отсчитать вверх или вниз, чтобы верхняя левая ячейка результата ссылалась на это место. Если задать, например, число 5 в качестве значения аргумента `смещ_по_строкам`, то тем самым вы укажете, что левая верхняя ячейка возвращаемой ссылки должна быть на пять строк ниже аргумента ссылка. `Смещ_по_строкам` может быть положительным (ниже начальной ссылки) или отрицательным (выше начальной ссылки).
- `Смещ_по_столбцам` — количество столбцов, которые требуется отсчитать влево или вправо, чтобы левая верхняя ячейка результата ссылалась на это место. Если задать, например, число 5 в качестве значения аргумента `смещ_по_столбцам`, то тем самым вы укажете, что левая верхняя ячейка возвращаемой ссылки должна быть на пять столбцов правее аргумента ссылка. `Смещ_по_столбцам` может быть положительным (вправо от начальной ссылки) или отрицательным (влево от начальной ссылки).
- `Высота` — высота (число строк) возвращаемой ссылки. Высота должна быть положительным числом.
- `Ширина` — ширина (число столбцов) возвращаемой ссылки. Ширина должна быть положительным числом.

НА ЗАМЕТКУ: Если в диапазоне данных используются типы данных, отличные от оговоренных, то функция `СЧЕТЗ` возвратит неправильный результат. Поэтому не стоит вводить в столбцы `А` и `В` данные с типами, отличными от числового и даты.

Именованная формула Продажи имеет следующий вид.

```
=СМЕЩ(Лист1!$B$2;0;0;СЧЕТЗ(Лист1!$B:$B)-1;1)
```

Если в столбце `В` введено 11 записей, то функция `СЧЕТЗ` возвратит значение 11. При расчете учитывается тот факт, что первая строка отводится под заголовки данных. В простом виде формула примет следующий вид.

```
=СМЕЩ(Лист1!$B$2;0;0;10;1)
```

В этой формуле ячейка `В2` используется в качестве точки привязки, относительно которой рассчитывается смещение.

- Смещение относительно ячейки `В2` на 0 строк.
- Смещение относительно ячейки `В2` на 0 столбцов.

- Диапазон, высотой 10 ячеек.
- Диапазон, шириной 1 ячейка.

Другими словами, функция `СМЕЩ` возвращает ссылку на диапазон `B2:B11`, на основе которого строится диаграмма. При добавлении еще одной записи функция `СМЕЩ` будет возвращать ссылку на диапазон `B2:B12`.

НА ЗАМЕТКУ

Чтобы упростить задачу, в данном примере использовался диапазон, состоящий из смежных ячеек. Тем не менее, эта методика применима и по отношению к комплексному диапазону, состоящему из нескольких областей рабочего листа.

Создание интерактивной диаграммы

В последнем примере этой главы, показанном на рис. 18.27, мы поговорим о приложении, которое позволяет отображать для выбранного города США (всего 284 города) диаграмму средней ежемесячной температуры, нормы осадков, средней скорости ветра и средней интенсивности солнечного освещения.

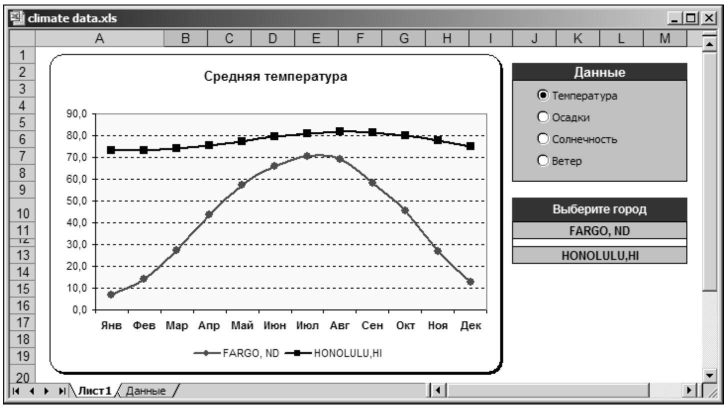


Рис. 18.27. В этом приложении используется несколько способов построения графиков изменения погодных условий

Интерактивность приложения обеспечивается встроенными средствами Excel — в нем не используются макросы. Город выбирается в раскрывающемся списке, он проверяется на принадлежность к списку, после чего указывается тип графика с помощью соответствующего переключателя. Работа приложения сводится к выполнению всего нескольких формул.

В этом примере показано, что удобное и практичное приложение можно создать без использования макросов.

НА КОМПАКТ-ДИСКЕ

Эта рабочая книга доступна на прилагаемом к книге компакт-диске.

В следующих подразделах поэтапно рассмотрены все операции по разработке этого приложения.

ПОЛУЧЕНИЕ ДАННЫХ ПРИЛОЖЕНИЯ

Всего несколько минут у меня ушло на поиск в Web исходных данных для создаваемого приложения. Я попросту скопировал данные из окна браузера в рабочий лист Excel, после чего немного упорядочил и отсортировал полученную информацию. В результате я получил таблицу данных, состоящую из 13 столбцов. Чтобы не загромождать экран данными, я разместил исходные данные на отдельном рабочем листе.

СОЗДАНИЕ НА РАБОЧЕМ ЛИСТЕ ПЕРЕКЛЮЧАТЕЛЕЙ

Нам необходимо позволить пользователям выбрать тип графика, который будет отображаться на экране. Один из лучших вариантов — это добавление элементов управления Переключатель с панели инструментов Формы. Поскольку переключатели объединяются в группы, то все они ссылаются на одну ячейку `O3`. Эта ячейка, тем не менее, может принимать несколько значений в диапазоне от 1 до 4 в зависимости от активного переключателя.

Далее необходимо реализовать механизм выбора таблицы данных на основе активного переключателя или значения в ячейке `O3`. В результате поиска решения мне удалось написать формулу, вводимую в ячейку `O4`, в которой используется функция `ВЫБОР`.

```
=ВЫБОР(O3;"TemperatureData";"PrecipitationData";"SunshineData";"WinData")
```

Таким образом, в ячейке `O4` отображается название одного из представляемых на диаграмме диапазонов данных. Для визуального выделения переключателей на рабочем листе я немного изменил форматирование прилегающих ячеек.

СОЗДАНИЕ СПИСКА ГОРОДОВ

Следующий этап — создание раскрывающегося списка, в котором пользователь сможет выбрать город с отображаемой для него информацией о погоде. Упрощает эту процедуру специальное средство проверки данных Excel. Но сначала нам придется выполнить предварительную обработку данных. Мы объединим ячейки `J11:M11` для первого города и дадим им название `City1`. Далее объединим ячейки `J13:M13` и дадим им название `City2`.

Для упрощения дальнейших действий я создал именованный диапазон `CityList`, который соответствует первому столбцу таблицы `PrecipitationData`.

Выполните приведенные ниже инструкции для создания раскрывающегося списка.

1. Выберите диапазон `J11:M11` (помните, что это объединенные ячейки).
2. Выполните команду Данные⇨Проверка. На экране появится диалоговое окно Проверка вводимых значений.
3. Перейдите на вкладку Параметры.
4. В поле Типы данных выберите значение Список.
5. В поле Источник введите `=CityList`.
6. Щелкните на кнопке OK.
7. Скопируйте диапазон `J11:M11` в диапазон `J13:M13`.

На рис. 18.28 показан полученный результат.

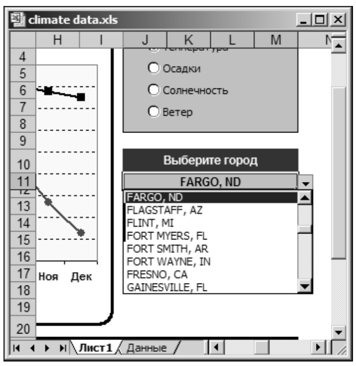
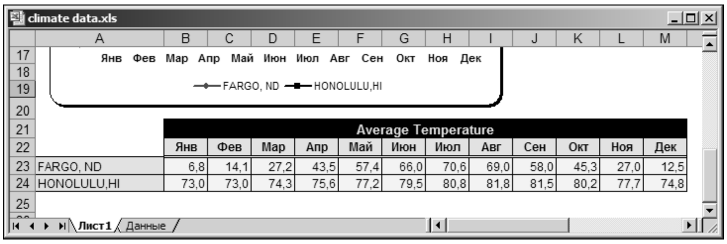


Рис. 18.28. Для выбора города используется раскрывающийся список

СОЗДАНИЯ ДИАПАЗОНА ДАННЫХ ДЛЯ ИНТЕРАКТИВНОЙ ДИАГРАММЫ

Ключевой момент в этой диаграмме заключается в следующем: диаграмма стоится на основе определенного диапазона данных. Данные в этом диапазоне получены из соответствующей таблицы данных с помощью формул, в которых фигурирует функция `ВПР`. На рис. 18.29 показан диапазон данных — диапазон, используемый при построении диаграммы.



	Average Temperature											
	Янв	Фев	Мар	Апр	Май	Июн	Июл	Авг	Сен	Окт	Ноя	Дек
FARGO, ND	6,8	14,1	27,2	43,5	57,4	66,0	70,6	69,0	58,0	45,3	27,0	12,5
HONOLULU,HI	73,0	73,0	74,3	75,6	77,2	79,5	80,8	81,8	81,5	80,2	77,7	74,8

Рис. 18.29. На диаграмме отображаются данные, полученные из диапазона `A22:M24`

В ячейке `A23` расположена формула, которая просматривает данные на основе содержимого ячейки `City1`.

```
=ВПР(City1;ДВССЫЛ(DataTable);СТОЛБЕЦ();ЛОЖЬ)
```

В ячейке `A24` находится формула, которая просматривает данные на основе содержимого ячейки `City2`.

```
=ВПР(City2;ДВССЫЛ(DataTable);СТОЛБЕЦ();ЛОЖЬ)
```

После введения этих формул я просто скопировал их в остальные 12 столбцов.

НА ЗАМЕТКУ

Вы можете быть удивлены тем фактом, что в третьем аргументе функции `ВПР` используется функция `СТОЛБЕЦ`. Эта функция возвращает номер столбца для ячейки ее аргумента. Тем самым обеспечивается простой способ указания рабочего столбца.

Подпись над месяцем генерируется с помощью формулы следующим образом.

```
="Средняя " &ЛЕВСИМВ(DataTable;ДЛСТР(DataTable)-4)
```

СОЗДАНИЕ ИНТЕРАКТИВНОЙ ДИАГРАММЫ

После выполнения всех описанных выше задач компоновка интерактивной диаграммы не должна вызывать больших затруднений. Финальный результат будет представлен двумя графиками, данные которых занесены в диапазон `A22:M24`. Заголовок диаграммы сохранен в ячейке `A21`. Данные в диапазоне `A22:M24` изменяются в зависимости от выбранного переключателя и города в раскрывающемся списке.

Глава 19

Концепция событий Excel

В ЭТОЙ ГЛАВЕ...

В этой главе рассматриваются концепции событий Excel и приводится ряд примеров, которые, без сомнения, пригодятся в вашей работе. Изучение методов, используемых в процессе обработки событий, позволяет добиться от приложения Excel дополнительной гибкости.

- Обзор событий, поддерживаемых Excel.
- Наиболее важная информация, которая необходима для управления событиями.
- Примеры событий, поддерживаемых объектами `Worksheet`, `Chart` и `UserForm`.
- Использование событий объекта `Application` для контроля за всеми открытыми рабочими книгами.
- Примеры обработки событий клавиатуры и времени.

В нескольких предыдущих главах этой книги рассматривались примеры процедур VBA, предназначенных для обработки событий. *Процедура обработки события* — это специально именованная процедура, которая запускается при возникновении определенного события. Простым примером может считаться процедура `CommandButton1_Click`, которая выполняется каждый раз, когда пользователь щелкает на элементе управления `CommandButton`, находящемся в пользовательском диалоговом окне или непосредственно на рабочем листе.

Excel контролирует возникновение широкого диапазона событий. При появлении определенного события Excel может запускать указанную процедуру обработки. Ниже приводится несколько примеров событий, распознаваемых программой Excel:

- открытие или закрытие рабочей книги;
- активизация окна;
- активизация или деактивизация листа;
- ввод данных в ячейку или редактирование данных в ячейке;
- сохранение рабочей книги;
- расчет значений в листе;
- щелчок на объекте;
- обновление данных на диаграмме;
- нажатие определенной клавиши или комбинации клавиш;
- двойной щелчок на ячейке;
- наступление определенного времени суток.

Типы событий Excel

Excel запрограммирована на управление большим количеством событий, которые происходят в процессе работы. Эти события могут быть классифицированы следующим образом.

- События объекта `Workbook` происходят в конкретной рабочей книге. Примером такого события можно назвать событие `Open` (возникает при открытии или создании рабочей книги), `BeforeSave` (возникает перед сохранением рабочей книги) и `NewSheet` (возникает при добавлении нового листа).
- События объекта `Worksheet` происходят в конкретном рабочем листе. В число примеров событий этого объекта можно включить `Change` (изменение содержимого ячейки на листе), `SelectionChange` (изменение расположения курсора) и `Calculate` (пересчет значений рабочего листа).
- События объекта `Chart` — к ним относятся события диаграммы. К таким событиям можно отнести `Select` (выделен объект на диаграмме) и `SeriesChange` (изменилось значение точки данных в последовательности). Для управления событиями встроенной диаграммы необходимо использовать модуль класса (см. главу 18).
- События объекта `Application` происходят в приложении Excel. В число таких событий входят `NewWorkbook` (создана новая рабочая книга), `WorkbookBeforeClose` (закрывается одна из рабочий книг) и `SheetChange` (изменено содержимое ячейки в одной из рабочих книг). Для контроля над событиями объекта `Application` необходимо использовать модуль класса.
- События объекта `UserForm`. Такие события происходят в определенном диалоговом окне `UserForm` или одном из объектов этого диалогового окна. Например, объект `UserForm` имеет событие `Initialize` (которое возникает перед отображением диалогового окна `UserForm`). Элемент управления `CommandButton`, который расположен в диалоговом окне `UserForm`, поддерживает событие `Click` (которое возникает при щелчке на этой кнопке).
- События, не связанные с объектами. Последняя категория состоит из таких событий уровня приложения (объекта `Application`), которые называются событиями “On-”: `OnTime` и `OnKey`. Эти события работают иначе, чем все остальные.

Данная глава организована в соответствии с приведенным выше списком. В пределах каждого из разделов приводятся примеры, которые демонстрируют использование некоторых событий.

Что необходимо знать о событиях

В этом разделе изложена важная информация, которая необходима для работы с событиями и в процессе создания процедур обработки событий.

Понимание последовательностей событий

Некоторые действия могут приводить к возникновению нескольких событий. Например, при добавлении нового листа в рабочую книгу возникают три события на уровне объекта `Application`.

- `WorkbookNewSheet`: происходит при добавлении нового листа.
- `SheetDeactivate`: происходит, когда деактивизируется активный лист.
- `SheetActivate`: происходит, когда активизируется добавленный лист.



Последовательность событий может быть сложнее, чем кажется. Перечисленные выше события происходят на уровне объекта `Application`. При добавлении нового листа возникают и дополнительные события (на уровне объектов `Workbook` и `Worksheet`).

Следует помнить, что события возникают в определенной последовательности. Знание порядка этой последовательности может оказаться очень важным при создании процедур обработки событий. Далее в этой главе будет рассматриваться порядок событий, возникающих при определенных действиях (дополнительна информация приведена в разделе “Отслеживание событий уровня объекта `Application`”).

Размещение процедур обработки событий

Начинающие разработчики VBA-приложений часто интересуются, почему их процедуры обработки событий не запускаются при возникновении соответствующего события. Ответ чаще всего заключается в том, что процедуры расположены в неправильном месте.

В окне редактора VBE все проекты перечислены в окне Projects. Компоненты проектов расположены в списке, показанном на рис. 19.1.

Все приведенные ниже компоненты имеют собственный модуль кода:

- объект `Лист` (например, `Лист1`, `Лист2` и т.д.);
- объект `Диаграмма` (т.е. листы диаграмм);
- объект `ЭтаКнига` (`ThisWorkbook`);
- модули VBA общего назначения: процедуру обработки события никогда нельзя размещать в модуле общего назначения (необъектном модуле);
- модули классов.

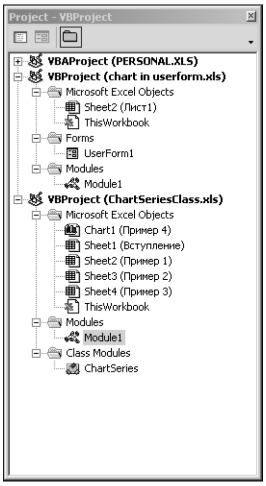


Рис. 19.1. Компоненты каждого проекта VBA перечислены в окне Projects

Даже если процедуры обработки событий располагаются в модулях объектов, они могут вызывать процедуры в модулях общего назначения для выполнения определенных задач. Например, следующая процедура обработки события, расположенная в модуле объекта `ЭтаКнига`, вызывает процедуру `WorkbookSetup`, которая может храниться в модуле кода VBA общего назначения.

```
Private Sub Workbook_Open()
    Call WorkbookSetup
End Sub
```

Отключение событий

По умолчанию все события включены. Для того чтобы предотвратить возникновение всех событий, выполните следующий оператор VBA.

```
Application.EnableEvents = False
```

Для разрешения возникновения событий воспользуйтесь таким оператором.

```
Application.EnableEvents = True
```

НА ЗАМЕТКУ: Предотвращение возникновения событий не влияет на события, которые проявляются в процессе использования элементов управления в диалоговом окне `UserForm`. Примером такого события может служить событие `Click`, которое генерируется при щелчке на элементе управления `CommandButton` в диалоговом окне `UserForm`.

Итак, зачем же отключать события? Основная причина заключается в необходимости предотвратить бесконечный цикл возникающих событий.

Например, ячейка `A1` на рабочем листе всегда должна содержать значение, которое меньше или равно 12. Можно написать процедуру, которая будет выполняться при каждом изменении данных в ячейке, что позволит проверять действительность содержимого этой ячейки. В данном случае контролируется событие `Change` объекта `Worksheet`, для чего используется процедура, которая называется `Worksheet_Change`. Процедура проверяет введенные пользователем данные, и если значение больше 12, то отображается окно сообщения, а содержимое ячейки очищается. Проблема заключается в том, что очистка содержимого ячейки с помощью кода VBA провоцирует возникновение еще одного события `Change`, что приводит к повторному вызову процедуры обработки события. Так как подобный результат нежелателен, перед очисткой содержимого ячейки необходимо отключить механизм возникновения событий. После выполнения операции включите механизм поддержки событий, чтобы иметь возможность контролировать следующую попытку пользователя ввести правильное значение в ячейку.

Еще одним способом предотвращения бесконечного цикла возникающих событий является декларирование статической булевой переменной (`Static Boolean`) в начале процедуры обработки события. Воспользуйтесь следующим выражением.

```
Static AbortProc As Boolean
```

Программирование событий в предыдущих версиях Excel

Программы Excel до версии 97 также поддерживают события, но методы программирования процедур обработки в них отличаются от описанных в этой главе.

Например, если вы создали процедуру `Auto_Open`, которая хранится в модуле VBA общего назначения, то эта процедура будет запускаться при каждом открытии рабочей книги. Начиная с Excel 97, процедура `Auto_Open` дополняется обработчиком события `Workbook_Open`, который хранится в модуле кода для объекта `ЭтаКнига`. Такой обработчик выполняется перед вызовом процедуры `Auto_Open`.

До выхода Excel 97 разработчикам приходилось часто явно настраивать события. Например, для запуска процедуры каждый раз, когда в ячейку вводились данные, необходимо было выполнять следующий оператор.

```
Sheets("Лист1").OnEntry = "ValidateEntry"
```

Этот оператор сообщает Excel, что следует запускать процедуру `ValidateEntry` каждый раз, когда данные вводятся в ячейку. В Excel 97 и более поздних версиях достаточно создать процедуру, которая называется `Worksheet_Change`, и сохранить ее в модуле кода для объекта `Лист1`.

По причине совместимости Excel 97 и более поздние версии программы поддерживают старый механизм обработки событий (хотя он и не документирован в справочном руководстве). Если вами разрабатываются приложения, которые будут использоваться в Excel 97 и более поздних версиях программы, то можно смело применять методы, описанные в этой главе.

Каждый раз, когда процедура должна внести изменения, необходимо устанавливать значение переменной `AbortProc` в значение `True` (в противном случае потребуется убедиться, что переменная установлена в значение `False`). Вставьте следующий код в начало процедуры.

```
If AbortProc Then
   AbortProc = False
   Exit Sub
End if
```

Если происходит повторный вызов процедуры обработки события, то значение `True` переменной `AbortProc` сообщает приложению о необходимости завершения этого вызова. Кроме того, значение переменной `AbortProc` устанавливается в значение `False`.

В качестве практического примера макроса проверки правильности введенных данных можно рассмотреть процедуру, приведенную в разделе "Проверка правильности введенных данных" далее в этой главе.

Отключение событий в Excel выполняется одновременно по отношению ко всем рабочим книгам. Например, если отключить события в одной процедуре и открыть другую рабочую книгу, в которой существует процедура `Workbook_Open`, то эта процедура также не будет запущена.

Ввод кода процедуры обработки события

Каждая процедура обработки события имеет свое предопределенное имя. Ниже приводятся названия отдельных процедур обработки событий.

- `Worksheet_SelectionChange`
- `Workbook_Open`
- `Chart_Activate`
- `Class_Initialize`

Можно объявить процедуру обработки события вручную, но целесообразнее возложить эту задачу на редактор VBE.

На рис. 19.2 показан модуль кода объекта `ЭтаКнига`. Для того чтобы добавить объявление процедуры, выберите объект `Workbook` из списка объектов слева. Затем выберите событие из списка процедур справа. После этого будет создана "оболочка" процедуры, которая состоит из строки декларации и оператора `End Sub`.

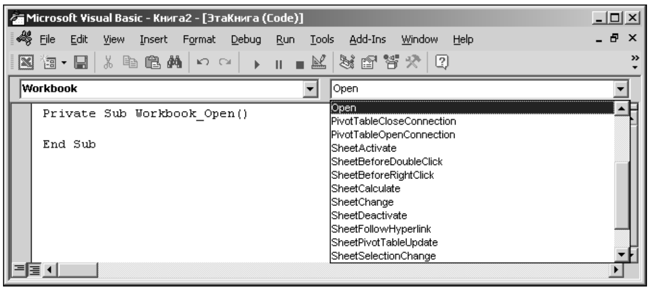


Рис. 19.2. Лучшим способом создания процедуры обработки события является использование встроенных средств VBE

Например, если выбрать из списка объектов `Workbook`, а из списка процедур — `Open`, то редактор VBE вставит приведенную ниже (пустую) процедуру:

```
Private Sub Workbook_Open()

End Sub
```

Создаваемый код должен помещаться между этими двумя строками.

Процедуры обработки событий, которые используют аргументы

Некоторые процедуры обработки событий используют набор аргументов. Например, может возникнуть необходимость в создании процедуры для обработки события `SheetActivate` рабочей книги. Если воспользоваться методикой, описанной в предыдущем разделе, то редактор VBE создаст представленную далее процедуру.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)

End Sub
```

Эта процедура использует один аргумент (`Sh`), который представляет активизированный лист. В данном случае переменная `Sh` имеет тип `Object`, а не просто тип `Worksheet`. Это связано с тем, что активизированный лист может оказаться листом диаграммы.

Данные, которые переданы в виде аргумента, могут быть применены в коде процедуры. В приведенном ниже примере процедура выполняется при активизации рабочего листа. На экране отображается тип и имя активизированного листа с помощью функции `TypeName` и свойства `Name` объекта, который передан в качестве аргумента.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    MsgBox TypeName(Sh) & vbCrLf & Sh.Name
End Sub
```

Некоторые процедуры обработки событий используют аргумент `Cancel` с типом данных `Boolean`. Например, объявление процедуры обработки события `BeforePrint` рабочей книги будет выглядеть следующим образом.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
```

Значение аргумента `Cancel`, которое передается в процедуру, равно `False`. Но код может установить это значение равным `True`, что приведет к отмене печати. Следующий пример демонстрирует такую операцию.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    Msg = "Вы загрузили необходимую бумагу?"
    Ans = MsgBox(Msg, vbYesNo, "В процессе печати...")
    If Ans = vbNo Then Cancel = True
End Sub
```

Процедура `Workbook_BeforePrint` выполняется перед тем, как печатается рабочая книга. Эта процедура отображает окно сообщения, которое показано на рис. 19.3. Если пользователь щелкнет на кнопке Нет, то аргумент `Cancel` приобретет значение `True`, и ничего напечатано не будет.

Событие `BeforePrint` возникает и перед предварительным просмотром рабочей книги.

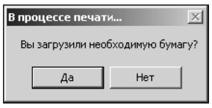


Рис. 19.3. Можно отменить операцию печати, изменив значение аргумента `Cancel`

К сожалению, Excel не обрабатывает событие `BeforePrint` на уровне листа. Таким образом, код не может определить, какая часть рабочей книги будет распечатана.

События объекта Workbook

События объекта `Workbook` происходят в пределах определенной рабочей книги. В табл. 19.1 перечислены события рабочей книги, а также приведено краткое описание каждого из них. Процедуры обработки событий объекта `Workbook` хранятся в модуле кода объекта `ThisWorkbook (ЭтаКнига)`.

Таблица 19.1. События объекта Workbook

Событие	Действие, которое приводит к возникновению этого события
`Activate`	Активизация рабочей книги
`AddinInstall`	Установка рабочей книги в качестве надстройки
`AddinUninstall`	Удаление рабочей книги, используемой в качестве надстройки
`BeforeClose`	Начало закрытия рабочей книги
`BeforePrint`	Начало процесса печати или предварительного просмотра рабочей книги или любого из ее элементов
`BeforeSave`	Начало сохранения рабочей книги
`Deactivate`	Деактивизация рабочей книги
`NewSheet`	Создание нового листа в рабочей книге
`Open`	Открытие рабочей книги
`PivotTableCloseConnection*`	Соединение с внешним источником данных сводной таблицы закрыто
`PivotTableOpenConnection*`	Соединение с внешним источником данных сводной таблицы открыто

* *Это событие происходит только в Excel 2002 и не поддерживается в предыдущих версиях.*

Событие	Действие, которое приводит к возникновению этого события
SheetActivate	Активизация любого листа
SheetBeforeDoubleClick	Двойной щелчок на любом из листов. Это событие происходит до принятого по умолчанию события двойного щелчка
SheetBeforeRightClick	Щелчок правой кнопкой мыши на любом листе. Это событие происходит до принятого по умолчанию события щелчка правой кнопкой мыши
SheetCalculate	Расчет (или перерасчет) значений в любом листе
SheetChange	Изменение любого листа пользователем или внешней ссылкой
SheetDeactivate	Деактивизация любого листа
SheetFollowHyperlink	Щелчок на гиперссылке в листе
SheetPivotTableUpdate	Пополнение сводной таблицы новыми данными
SheetSelectionChange	Изменение выделения на любом листе
WindowActivate	Активизация окна любой рабочей книги
WindowDeactivate	Деактивизация окна любой рабочей книги
WindowResize	Изменение размера окна любой рабочей книги

Если необходимо проследить за событиями, возникающими для всех рабочих книг, то придется работать на уровне объекта Application (см. раздел "События объекта Application" далее в этой главе). Ниже приведены примеры использования событий объекта Workbook. Все примеры процедур, представленные ниже, должны располагаются в модуле кода объекта ЭтаКнига. Если разместить их в модуле кода другого типа, то они не будут работать.

Событие Open

Одним из наиболее часто используемых является событие Open объекта рабочей книги. Это событие возникает при открытии рабочей книги (или надстройки). При этом выполняется процедура Workbook_Open. Процедура Workbook_Open может выполнять любые действия, но чаще всего она используется для решения следующих задач:

- отображения приветственных сообщений;
- открытия других рабочих книг;
- настройки пользовательских меню или панелей инструментов;
- активизации определенного листа или ячейки;
- проверки определенных условий (например, рабочая книга может требовать наличия определенной надстройки);
- установки определенных автоматических средств (например, можно определять комбинации клавиш — см. раздел "Событие OnKey" далее в этой главе);
- настройки свойства ScrollArea рабочего листа (которое не хранится в рабочей книге);
- настройки защиты UserInterfaceOnly для рабочего листа, что позволяет управлять защищенными листами в коде. Этот параметр является аргументом метода Protect и не хранится в пределах рабочей книги.

Если пользователь будет удерживать клавишу <Shift> в момент открытия рабочей книги, то процедура `Workbook_Open` не сможет быть запущена.

Ниже приводится простой пример процедуры `Workbook_Open`. Эта процедура использует функцию `Weekday` для определения дня недели. Если текущий день недели пятница, то отображается окно сообщения, которое напоминает пользователю о необходимости проведения еженедельного резервного копирования файлов. Если текущий день недели не пятница, то ничего не происходит.

```
Private Sub Workbook_Open()
  If Weekday(Now) = vbFriday Then
      Msg = "Сегодня пятница. Ты уже создал"
      Msg = Msg & "новую резервную копию?"
      MsgBox Msg, vbInformation
  End If
End Sub
```

Событие Activate

Следующая процедура выполняется при каждой активизации рабочей книги. Эта процедура разворачивает активное окно.

```
Private Sub Workbook_Activate()
    ActiveWindow.WindowState = xlMaximized
End Sub
```

Событие SheetActivate

Представленная далее процедура выполняется каждый раз, когда пользователь активизирует один из листов рабочей книги. Если этот лист является рабочим листом, то выделяется ячейка `A1`. Если он не является рабочим листом, то ничего не происходит. Данная процедура использует функцию VBA `TypeName` для того, чтобы удостовериться, что активный лист является рабочим листом (не представлен листом диаграммы).

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    If TypeName(Sh) = "Worksheet" Then _
       Range("A1").Select
End Sub
```

Существует и другой метод избежать ошибки при выделении ячейки на листе диаграммы — просто игнорируйте такую ошибку.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    On Error Resume Next
    Range("A1").Select
End Sub
```

Событие NewSheet

Рассматриваемая процедура выполняется каждый раз, когда к рабочей книге добавляется лист. Лист передается в процедуру в качестве аргумента. Так как новый лист может выступать как рабочим листом, так и листом диаграммы, то эта процедура в обязательном порядке определяет тип листа. Если вставляется рабочий лист, то код создает запись о дате и времени добавления листа в ячейку `A1`.

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
    If TypeName(Sh) = "Worksheet" Then _
      Range("A1") = "Лист добавлен " & Now()
End Sub
```

Событие BeforeSave

Событие `BeforeSave` возникает перед фактическим сохранением рабочей книги. Как известно, использование команды Файл⇨Сохранить приводит к появлению диалогового окна Сохранение документа. Это происходит в том случае, если рабочая книга еще ни разу не сохранялась и не открывалась в режиме "только чтения".

При выполнении процедуры `Workbook_BeforeSave` ей передается аргумент (`SaveAsUI`), который позволяет предвосхитить появление диалогового окна Сохранение документа. Следующий пример демонстрирует использование этого события.

```
Private Sub Workbook_BeforeSave _
  (ByVal SaveAsUI As Boolean, Cancel As Boolean)
    If SaveAsUI Then
        MsgBox "Щелкните на кнопке OK для сохранения."
    End If
End Sub
```

Когда пользователь сохраняет рабочую книгу, выполняется процедура `Workbook_BeforeSave`. Если операция сохранения приводит к появлению диалогового окна Сохранение документа, то переменной `SaveAsUI` присваивается значение `True`. Процедура проверяет переменную `SaveAsUI`; в результате отображается сообщение лишь в том случае, если на экране появляется окно сохранения документа. При установлении процедурой аргумента `Cancel` в значение `True` файл сохранен не будет.

Событие Deactivate

Предлагаемый пример демонстрирует использование события `Deactivate`. Эта процедура выполняется каждый раз при деактивизации рабочей книги. Она не позволяет пользователю деактивизировать рабочую книгу. При возникновении события `Deactivate` код производит повторную активизацию рабочей книги и выводит на экран соответствующее сообщение.

```
Private Sub Workbook_Deactivate()
    Me.Windows(1).Activate
    MsgBox "Мне жаль, но вы не покинете эту книгу!"
End Sub
```

НА ЗАМЕТКУ

Использовать такие процедуры не рекомендуется. Если процедура пытается "взвалить" на себя задачи Excel, то ее действия могут привести пользователя в замешательство. Мы настоятельно рекомендуем обучать пользователей правильной работе с приложением.

Приводимый пример иллюстрирует важность обработки событий в необходимой последовательности. Запустив эту процедуру, вы убедитесь, что она работает правильно только в тех случаях, когда пользователь активизирует другую рабочую книгу. Событие `Deactivate` для рабочей книги возникает и в ряде других случаев:

- при закрытии рабочей книги;
- при открытии новой рабочей книги;
- при сворачивании окна рабочей книги.

Таким образом, данная процедура не будет работать точно так, как это задумывалось изначально. Она будет предотвращать попытки пользователя непосредственно активизировать другую рабочую книгу, но не сможет воспрепятствовать закрытию рабочей книги, открытию новой рабочей книги или сворачиванию окна рабочей книги. Окно сообщения будет отображаться, но действия все же будут возможны.

Событие BeforePrint

Событие `BeforePrint` происходит, когда пользователь отправляет документ на печать или отображает его (или фрагмент рабочей книги) в режиме предварительного просмотра. Данное событие происходит перед началом фактической печати или предварительного просмотра рабочей книги. Процедура обработки этого события принимает аргумент `Cancel`, чтобы в коде можно было отменить печать или выполнить предварительный просмотр в результате установки этого аргумента в значение `True`. К сожалению, не существует способа определить, чем было спровоцировано событие `BeforePrint`: отправкой задания печати или запросом предварительного просмотра.

В Excel до версии 2002 нельзя было в верхнем или нижнем колонтитуле страницы напечатать полный путь к файлу, в котором хранилась рабочая книга. Excel 2002 решает эту проблему. Теперь на вкладке Колонтитулы (доступ к ней вы получите в диалоговом окне Параметры страницы) можно определить все необходимые данные колонтитулов.

Если вы используете старую версию Excel, то единственным решением будет создание кода, который добавляет путь к рабочей книге в верхний или нижний колонтитул. Процедура обработки события `BeforePrint` является идеальным способом размещения такого кода. Процедура `Workbook_BeforePrint`, содержащая подобный код, приводится ниже.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    For Each sht In ThisWorkbook.Sheets
        sht.PageSetup.LeftFooter = _
          "&8" & ThisWorkbook.FullName
    Next sht
End Sub
```

Эта процедура просматривает все листы рабочей книги и присваивает свойству `LeftFooter` объекта `PageSetup` значение свойства `FileName` объекта рабочей книги (которое содержит путь и имя файла, где хранится рабочая книга). Кроме того, размер шрифта устанавливается в значение 8 пунктов.

Приведенный пример демонстрирует неоднородность объектной модели Excel. Для того чтобы изменить размер шрифта нижнего и верхнего колонтитулов, необходимо применить специальный код форматирования. В примере, показанном выше, используется строка `"&8"` для назначения шрифта размером 8 пунктов. В идеале в объектах колонтитулов должен использоваться объект `Font`. Чтобы определить доступность других способов форматирования, необходимо обратиться к справочному руководству (или записать макрос в момент доступа к диалоговому окну Параметры страницы, а затем проанализировать полученный код).

При тестировании процедуры обработки события `BeforePrint` вы сможете сэкономить время (и бумагу), выполнив предварительный просмотр документа вместо отправки его на печать.

Событие BeforeClose

Событие `BeforeClose` происходит перед закрытием рабочей книги. Оно часто используется совместно с процедурой `Workbook_Open`. Например, можно использовать процедуру `Workbook_Open` для создания дополнительной опции меню, специфичной для конкретной рабочей книги, а процедуру `Workbook_BeforeClose` — для удаления этого дополнительного элемента из меню. Таким образом вы обеспечите доступность опции меню только в открытой рабочей книге.

Как известно, при попытке закрыть несохраненную рабочую книгу Excel выдает сообщение с запросом на сохранение рабочей книги перед ее закрытием (рис. 19.4).

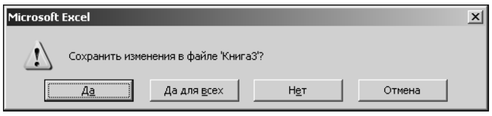


Рис. 19.4. Если отображается это сообщение, значит, процедура `Workbook_BeforeClose` *уже завершила свою работу*

Обратим ваше внимание на потенциальную проблему, которая заключается в следующем: к моменту отображения этого сообщения событие `BeforeClose` уже возникло, что указывает на завершение работы процедуры `Workbook_BeforeClose`.

Рассмотрим иной сценарий развития событий: в момент открытия определенной рабочей книги отображается пользовательское меню. В данном случае рабочая книга использует процедуру `Workbook_Open` для создания меню при открытии рабочей книги, а процедура `Workbook_BeforeClose` — для удаления этого меню при закрытии рабочей книги. Процедуры обработки событий приведены ниже. Обе процедуры вызывают другие процедуры, которые нами не рассматриваются.

```
Private Sub Workbook_Open()
    Call CreateMenu
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Call DeleteMenu
End Sub
```

Как отмечалось выше, сообщение Excel `Сохранить изменения...` отображается *после* того, как процедура `Workbook_BeforeClose` завершит свою работу. Поэтому если пользователь щелкнет на кнопке Отмена, то рабочая книга останется открытой, но дополнительный элемент меню будет уже удален!

Одним из решений подобной проблемы является создание кода запроса на сохранение в процедуре `Workbook_BeforeClose`. Ниже приведен пример, демонстрирующий способ выполнения подобной задачи.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Dim Msg As String
    If Me.Saved Then
        Call DeleteMenu
        Exit Sub
    Else
        Msg = "Сохранить изменения в документе?"
        Msg = Msg & Me.Name & "?"
        Ans = MsgBox(Msg, vbQuestion + vbYesNoCancel)
        Select Case Ans
            Case vbYes
                Me.Save
                Call DeleteMenu
            Case vbNo
                Me.Saved = True
                Call DeleteMenu
            Case vbCancel
                Cancel = True
        End Select
    End If
End Sub
```

Данная процедура проверяет значение свойства `Saved` объекта `Workbook` с целью определить наличие в рабочей книге несохраненных данных. Если они существуют, то выполняется процедура `DeleteMenu`, и рабочая книга закрывается. Однако в рабочей книге могут оставаться несохраненные данные. Тогда процедура отображает окно сообщения, которое дублирует исходное окно сообщения Excel. Если пользователь щелкнет на кнопке Да, то рабочая книга будет сохранена, элемент меню удален, а рабочая книга — закрыта. Если пользователь щелкнет на кнопке Нет, то код установит свойство `Saved` объекта `Workbook` в значение `True` (но не будет сохранять файл) и удалит опцию меню. Если пользователь щелкнет на кнопке Отмена, то событие `BeforeClose` будет отменено и процедура завершится без удаления элемента меню.

События объекта Worksheet

События объекта `Worksheet` являются самыми полезными и часто используемыми. Контроль над ними может заставить приложение выполнять функции, которые в другой ситуации считались бы невозможными.

События, представленные в этом разделе, относятся только к рабочим листам. Не существует подобных событий для диалоговых листов и листов макросов XLM в Excel 5/95.

Таблица 19.2. События объекта Worksheet

Событие	Действие, которое приводит к возникновению этого события
`Activate`	Активизация рабочего листа
`BeforeDoubleClick`	Двойной щелчок на рабочем листе
`BeforeRightClick`	Щелчок правой кнопкой мыши на рабочем листе
`Calculate`	Расчет (или перерасчет) значений рабочего листа
`Change`	Значение ячейки рабочего листа изменено пользователем или внешней ссылкой
`Deactivate`	Деактивизация рабочего листа
`FollowHyperlink`	Щелчок на гиперссылке в рабочем листе
`PivotTableUpdate*`	Обновление сводной таблицы на рабочем листе
`SelectionChange`	Перемещение курсора на рабочем листе

* *Это событие происходит только в Excel 2002 и не поддерживается в предыдущих версиях.*

Помните, что код процедуры обработки события рабочего листа должен храниться в модуле кода соответствующего объекта рабочего листа.

Для того чтобы быстро активизировать модуль кода для рабочего листа, щелкните правой кнопкой мыши на ярлыке листа и выберите команду Исходный текст.

Событие Change

Событие `Change` возникает в момент изменения значения ячейки пользователем или VBA-процедурой. Событие `Change` не возникает, когда расчеты приводят к появлению другого значения формулы или когда на рабочий лист добавляется новый объект.

При вызове процедуры Worksheet_Change в качестве аргумента Target ей передается объект Range. Этот объект представляет изменившуюся ячейку или диапазон, которые привели к возникновению события. Следующий пример отображает окно сообщения, которое выводит адрес диапазона, указанного в параметре Target.

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
    MsgBox "Диапазон " & Target.Address & " изменился"
End Sub
```

Для того чтобы получить представление о типах действий, которые приводят к возникновению события Change в рабочем листе, добавьте предыдущую процедуру в модуль кода объекта Worksheet. После ввода этой процедуры активизируйте Excel и внесите изменения в рабочий лист, используя для этого различные методы. Каждый раз при возникновении события Change будет отображаться адрес диапазона, который изменился.

После запуска этой процедуры вами может быть замечена интересная особенность: некоторые действия, которые должны способствовать возникновению этого события, ни к чему не приводят, а те, которые не должны выполнять эту задачу, приводят к возникновению события Change!

- Изменение форматирования ячейки не приводит к возникновению события Change, а использование команды Правка⇨Очистить⇨Форматы, наоборот, вызывает это событие.
- Добавление, редактирование или удаление комментария в ячейке не приводит к возникновению события Change.
- Нажатие клавиши <Del> приводит к генерации события Change даже в том случае, если ячейка не содержит данных.
- Ячейки, которые изменяются с помощью команд Excel, могут генерировать, а могут и не генерировать событие Change. Например, команды Данные⇨Форма и Данные⇨Сортировка не приводят к возникновению события Change. Зато команды Сервис⇨Орфография и Правка⇨Заменить генерируют это событие.
- Если процедура VBA изменяет содержимое ячейки, то это приводит к возникновению события Change.

Рассматривая приведенный выше список, вы убедитесь, что использовать событие Change для отслеживания изменений в ячейках нецелесообразно.

НА ЗАМЕТКУ

Кроме перечисленных выше особенностей, возникновение события Change в ответ на определенные действия зависит и от версии Excel. Во всех версиях Excel до 2002 заполнение диапазона с помощью команды Правка⇨Заполнить не приводило к возникновению события Change. Подобным образом вела себя команда Правка⇨Удалить, которая удаляла содержимое выделенного диапазона.

Отслеживание изменений в определенном диапазоне

Событие Change возникает при внесении изменений в любую ячейку рабочего листа. Но в большинстве случаев важно отслеживать изменения, которые вносятся в определенную ячейку или диапазон. Когда вызывается процедура обработки события Worksheet_Change, она получает в качестве параметра объект Range. Этот объект представляет диапазон ячеек или ячейку, которая после изменения приводит к возникновению события Change. Предположим, что на рабочем листе определен диапазон InputRange, и вам необходимо отслеживать только те изменения, которые внесены

в этом диапазоне. Не существует события Change для отдельного объекта Range, но можно выполнить необходимую проверку в начале процедуры Worksheet_Change.

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
    Dim VRange As Range
    Set VRange = Range("InputRange")
    If Not Intersect(Target, VRange) Is Nothing Then _
       MsgBox "Изменена ячейка текущего диапазона."
End Sub
```

В данном примере используется объект Range, который называется VRange. Он представляет диапазон ячеек на рабочем листе, который необходимо проверять на предмет внесения изменений. Процедура использует функцию VBA Intersect, чтобы определить расположение диапазона Target (полученного в качестве атрибута процедуры) в диапазоне VRange. Функция Intersect возвращает объект, который состоит из всех ячеек, содержащихся в обоих аргументах. Если функция Intersect возвращает значение Nothing, то эти диапазоны не имеют общих ячеек. Оператор отрицания Not используется для того, чтобы выражение стало равным True в том случае, если указанный диапазон имеет хотя бы одну общую ячейку с диапазоном VRange. Таким образом, будет отображено окно сообщения. В противном случае ничего не происходит, и процедура завершает свою работу.

ВНЕСЕНИЕ ЗАПИСЕЙ ОБ ИЗМЕНЕНИЯХ В КОММЕНТАРИИ ЯЧЕЕК

В следующем примере представлен процесс добавления заметок к комментарию в ячейке при внесении в нее изменений (что определяется событием Change). Значение элемента управления CheckBox, добавленного на лист, определяет необходимость внесения заметок в комментарий ячейки.

Этот пример содержится на прилагаемом к книге компакт-диске.

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
    Dim cell As Range
    Dim OldText As String, NewText As String

  If CheckBox1 Then
    For Each cell In Target
        With cell
            On Error Resume Next
            OldText = .Comment.Text
            If Err <> 0 Then .AddComment
            NewText = OldText & "Изменено на " & cell.Text & _
              ". " & Application.UserName & ", " & Now & vbLf
            .Comment.Text NewText
            .Comment.Visible = True
            .Comment.Shape.Select
             Selection.AutoSize = True
            .Comment.Visible = False
        End With
    Next cell
  End If
End Sub
```

Так как объект, который передан в процедуру Worksheet_Change, может состоять из многоячеечного диапазона, процедура циклически просматривает все ячейки диапазона Target. Если ячейка еще не содержит комментария, то он добавляется. Новый текст будет добавлен в конец уже существующего комментария (если такой существует).

ПРОВЕРКА ПРАВИЛЬНОСТИ ВВЕДЕННЫХ ДАННЫХ

Средство Excel проверки данных может оказаться очень ценным инструментом, но оно имеет серьезную потенциальную проблему. При вставке данных в ту ячейку, в которой реализуется проверка данных, значение не только не проверяется, но и правила проверки, которые связаны с этой ячейкой, безвозвратно удаляются! Таким образом, инструмент проверки данных становится практически бесполезным в собственных приложениях. В этом разделе продемонстрированы методы использования события `Change` объекта `Worksheet` для создания процедур проверки правильности введенных данных.

НА КОМПАКТ-ДИСКЕ

На прилагаемом компакт-диске содержится две версии этого примера. В одной из них используется свойство `EnableEvents` для предотвращения бесконечного цикла возникновения событий `Change`. Во второй версии применена статическая булева переменная (дополнительная информация об отключении событий была приведена в разделе "Отключение событий" ранее в этой главе).

Листинг 19.1 содержит процедуру, которая выполняется каждый раз при внесении изменений в ячейку со стороны пользователя. Проверка ограничена диапазоном, называющимся `InputRange`. В этот диапазон разрешается вводить только целые значения от 1 до 12.

Листинг 19.1. Определение правильности введенных данных

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
    Dim VRange As Range, cell As Range
    Dim Msg As String
    Dim ValidateCode As Variant
    Set VRange = Range("InputRange")
    For Each cell In Target
        If Union(cell, VRange).Address = VRange.Address Then
            ValidateCode = EntryIsValid(cell)
            If ValidateCode = True Then
                Exit Sub
            Else
                Msg = "Ячейка " & cell.Address(False, False) & ":"
                Msg = Msg & vbCrLf & vbCrLf & ValidateCode
                MsgBox Msg, vbCritical, "Неправильное значение"
                Application.EnableEvents = False
                cell.ClearContents
                cell.Activate
                Application.EnableEvents = True
            End If
        End If
    Next cell
End Sub
```

Процедура `Worksheet_Change` создает объект `Range`. Он называется `VRange` и представляет проверяемый диапазон на рабочем листе. В результате циклически просматриваются все ячейки аргумента `Target`, представляющего диапазон изменившихся ячеек. В коде определяется, находится ли изменившаяся ячейка в проверяемом диапазоне, и если это так, передает ячейку в качестве аргумента другой функции (`EntryIsValid`), возвращающей значение `True` для действительного значения ячейки.

Если значение ячейки выходит за пределы набора допустимых значений, то функция `EntryIsValid` возвращает строку, которая описывает проблему. Затем выводится окно сообщения (рис. 19.5). После закрытия окна сообщения неправильное значение ячейки удаляется, и ячейка активизируется. Обратите внимание, что перед удалением значения ячейки события отключаются. Если события не отключить, то ячейка создаст событие `Change` и введет процедуру в бесконечный цикл.

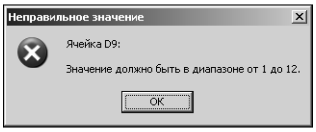


Рис. 19.5. Это окно сообщения описывает проблему, возникающую при введении пользователем недопустимых данных

Ниже приведен листинг процедуры `EntryIsValid`.

Листинг 19.2. Проверка принадлежности значения диапазону

```
Private Function EntryIsValid(cell) As Variant
'   Возвращает True, если в ячейку вводится целое число в диапазоне от 1 до 12
'   В противном случае возвращается строка, описывающая проблему

'   Число
    If Not WorksheetFunction.IsNumber(cell) Then
        EntryIsValid = "Нечисловое значение."
        Exit Function
    End If

'   Целое?
    If CInt(cell) <> cell Then
        EntryIsValid = "Введите целое число."
        Exit Function
    End If

'   Диапазон?
    If cell < 1 Or cell > 12 Then
        EntryIsValid = "Значение должно быть в диапазоне от 1 до 12."
        Exit Function
    End If

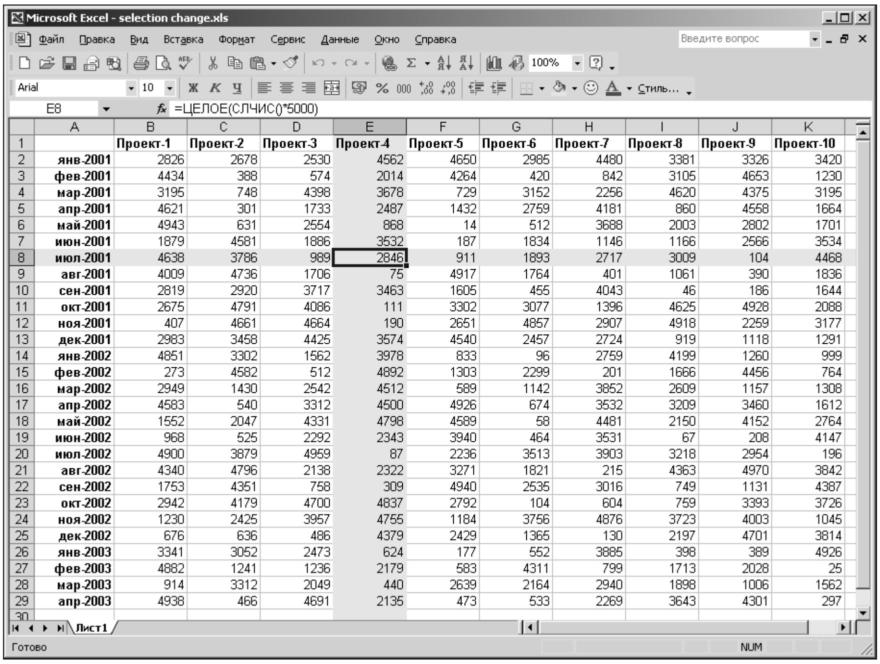
'   Тест завершен успешно
    EntryIsValid = True
End Function
```

Событие SelectionChange

Следующая процедура демонстрирует использование события `SelectionChange`. Она выполняется каждый раз, когда пользователь создает новое выделение на рабочем листе.

```
Private Sub Worksheet_SelectionChange(ByVal Target _
 As Excel.Range)
    Cells.Interior.ColorIndex = xlNone
    With ActiveCell
        .EntireRow.Interior.ColorIndex = 27
        .EntireColumn.Interior.ColorIndex = 27
    End With
End Sub
```

Данная процедура выделяет строку и столбец активной ячейки, что облегчает визуальный поиск последней. Первый оператор нейтрализует цвет фона для всех ячеек рабочего листа, после чего строка и столбец активной ячейки выделяются светло-желтым цветом. На рис. 19.6 отображен эффект выделения (поверьте, это желтый цвет).



	A	B	C	D	E	F	G	H	I	J	K
1		Проект-1	Проект-2	Проект-3	Проект-4	Проект-5	Проект-6	Проект-7	Проект-8	Проект-9	Проект-10
2	янв-2001	2826	2678	2530	4562	4650	2985	4480	3381	3326	3420
3	фев-2001	4434	388	574	2014	4264	420	842	3105	4653	1230
4	мар-2001	3195	748	4398	3678	729	3152	2256	4620	4375	3195
5	апр-2001	4621	301	1733	2487	1432	2759	4181	860	4558	1664
6	май-2001	4943	631	2554	868	14	512	3688	2003	2802	1701
7	июн-2001	1879	4581	1886	3532	187	1834	1146	1166	2566	3534
8	июл-2001	4638	3786	989	2846	911	1893	2717	3009	104	4468
9	авг-2001	4009	4736	1706	75	4917	1764	401	1061	390	1836
10	сен-2001	2819	2920	3717	3463	1605	455	4043	46	186	1644
11	окт-2001	2675	4791	4086	111	3302	3077	1396	4625	4928	2088
12	ноя-2001	407	4661	4664	190	2651	4857	2907	4918	2259	3177
13	дек-2001	2983	3458	4425	3574	4540	2457	2724	919	1118	1291
14	янв-2002	4851	3302	1562	3978	833	96	2759	4199	1260	999
15	фев-2002	273	4582	512	4892	1303	2299	201	1666	4456	764
16	мар-2002	2949	1430	2542	4512	589	1142	3852	2609	1157	1308
17	апр-2002	4583	540	3312	4500	4926	674	3532	3209	3460	1612
18	май-2002	1552	2047	4331	4798	4589	58	4481	2150	4152	2764
19	июн-2002	968	525	2292	2343	3940	464	3531	67	208	4147
20	июл-2002	4900	3879	4959	87	2236	3513	3903	3218	2954	196
21	авг-2002	4340	4796	2138	2322	3271	1821	215	4363	4970	3842
22	сен-2002	1753	4351	758	309	4940	2535	3016	749	1131	4387
23	окт-2002	2942	4179	4700	4837	2792	104	604	759	3393	3726
24	ноя-2002	1230	2425	3957	4755	1184	3756	4876	3723	4003	1045
25	дек-2002	676	636	486	4379	2429	1365	130	2197	4701	3814
26	янв-2003	3341	3052	2473	624	177	552	3885	398	389	4926
27	фев-2003	4882	1241	1236	2179	583	4311	799	1713	2028	25
28	мар-2003	914	3312	2049	440	2639	2164	2940	1898	1006	1562
29	апр-2003	4938	466	4691	2135	473	533	2269	3643	4301	297

Рис. 19.6. Щелчок на ячейке приводит к выделению ее строки и столбца

Этот пример доступен на прилагаемом к книге компакт-диске.

Не рекомендуется использовать эту процедуру, если рабочий лист содержит диапазоны, уже выделенные другими цветами. В результате выполнения этой процедуры выделение ячеек цветом удаляется.

Событие BeforeRightClick

Когда пользователь щелкает правой кнопкой мыши на рабочем листе, Excel отображает контекстное меню. Если по определенной причине необходимо отключить появление контекстного меню на одном из рабочих листов, то можно перехватить событие `BeforeRightClick`. Следующая процедура устанавливает значение аргумента `Cancel` равным `True`, что приводит к отмене обработки события `BeforeRightClick`. Таким образом, можно отключить появление контекстного меню в текущем рабочем листе.

```
Private Sub Worksheet_BeforeRightClick _
  (ByVal Target As Excel.Range, Cancel As Boolean)
    Cancel = True
    MsgBox "Контекстное меню не доступно."
End Sub
```

В главе 24 описываются другие методы отключения контекстного меню.

События объекта Chart

По умолчанию события разрешены только для диаграмм, которые находятся на листах диаграмм. Для того чтобы работать с событиями встроенных диаграмм, необходимо создать модуль класса.

Обратитесь к главе 18 за примерами управления событиями объекта `Chart`. Кроме того, в главе 18 описывается создание модуля класса для поддержки событий встроенных диаграмм.

В табл. 19.3 приведен список событий, поддерживаемых диаграммами, а также представлено краткое описание каждого события.

Таблица 19.3. События, которые поддерживаются листами диаграмм

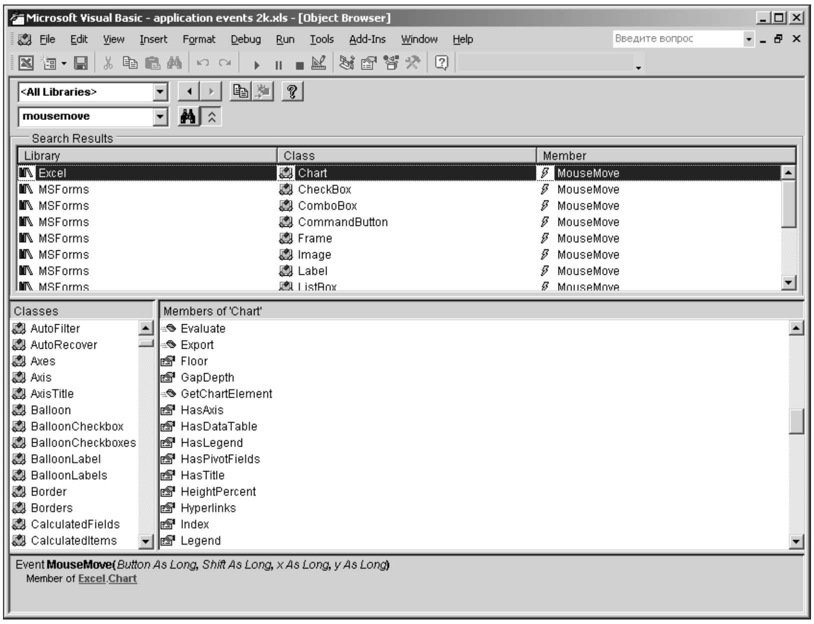
Событие	Действие, которое приводит к возникновению события
`Activate`	Активизация листа диаграммы или встроенной диаграммы
`BeforeDoubleClick`	Двойной щелчок на листе диаграммы или на встроенной диаграмме. Это событие происходит до обработки двойного щелчка, принятого по умолчанию
`BeforeRightClick`	Щелчок правой кнопкой мыши на листе диаграммы или на встроенной диаграмме. Это событие происходит до обработки такого щелчка правой кнопкой мыши, который принят по умолчанию
`Calculate`	На диаграмме отображаются новые или обновленные данные
`Deactivate`	Деактивизация диаграммы
`DragOver`	Диапазон ячеек перетаскивается через диаграмму
`DragPoint`	Диапазон ячеек перетаскивается и отпускается на диаграмму
`MouseDown`	Кнопка мыши нажимается в тот момент, когда указатель располагается над диаграммой
`MouseMove`	Указатель мыши перемещается над диаграммой
`MouseUp`	Кнопка мыши отпускается в тот момент, когда указатель располагается над диаграммой
`Resize`	Изменились размеры диаграммы
`Select`	Выделен один из элементов диаграммы
`SeriesChange`	Изменилось значение точки данных в одной из последовательностей диаграммы

События объекта Application

В предыдущих разделах были описаны события объектов `Workbook` и `Worksheet`. Эти события рассматривались по отношению к определенной рабочей книге или рабочему листу. Если требуется проконтролировать возникновение событий для всех открытых рабочих книг или рабочих листов, то рекомендую обратиться к событиям уровня объекта `Application`.

Использование браузера объектов для поиска событий

Окно Object Browser является полезным инструментом — оно помогает получить информацию об объектах, их свойствах и методах. Кроме того, окно Object Browser предоставляет возможность определить, какие объекты поддерживают определенное событие. Предположим, что необходимо найти объекты, которые поддерживают событие `MouseMove`. Активизируйте редактор VBE и нажмите клавишу <F2>, чтобы отобразить окно Object Browser. Удостоверьтесь, что выбран пункт <All Libraries>, после чего введите **MouseMove** и щелкните на кнопке с изображением бинокля.



Окно Object Browser отобразит список подходящих объектов. События обозначаются с помощью небольшой желтой молнии. На основании этого списка можно определить, какие объекты поддерживают событие `MouseMove`. Многие объекты, которые будут найдены, являются элементами управления, находящимися в библиотеке `MSForms`. Элемент управления `UserForm` также находится в этой библиотеке, а событие `MouseMove` поддерживается даже объектом `Chart`.

Обратите внимание на разделение списка на три столбца: Library, Class и Members. Элемент, соответствующий критерию поиска, может оказаться в любом из этих столбцов, что приводит к сложной проблеме: имя события или термина одновременно принадлежит двум библиотекам или классам. Но они не всегда предоставляют одинаковую функциональность. Поэтому следует щелкнуть на каждом из интересующих элементов, отображенных в окне Object Browser, и сравнить информацию, которая приводится под списком. Может оказаться, что отдельный класс или библиотека рассматривают событие совершенно по-другому.

НА ЗАМЕТКУ

Создание процедуры обработки события для объекта `Application` требует создания модуля класса и дополнительных действий по настройке среды.

В табл. 19.4 перечислены события объекта `Application` с кратким описанием каждого из них.

Таблица 19.4. События, поддерживаемые объектом Application

Событие	Действие, которое приводит к возникновению события
NewWorkbook	Создана новая рабочая книга
SheetActivate	Активизирован любой лист
SheetBeforeDoubleClick	Двойной щелчок на любом из листов. Это событие происходит до обработки двойного щелчка, принятого по умолчанию
SheetBeforeRightClick	Щелчок правой кнопкой мыши на любом из листов. Это событие происходит до обработки щелчка правой кнопкой мыши, принятого по умолчанию
SheetCalculate	Рассчитывается (или пересчитывается) любой лист
SheetChange	Ячейки в любом листе были изменены пользователем или со стороны внешней ссылки
SheetDeactivate	Деактивизирован любой лист
SheetFollowHyperlink	Щелчок на гиперссылке
SheetPivotTableUpdate*	Обновление сводной таблицы
SheetSelectionChange	Изменилось выделение на любом листе, кроме листа диаграммы
WindowActivate	Активизировано окно любой рабочей книги
WindowDeactivate	Деактивизировано окно любой рабочей книги
WindowResize	Изменены размеры окна любой рабочей книги
WorkbookActivate	Активизирована рабочая книга
WorkbookAddinInstall	Рабочая книга установлена в качестве надстройки
WorkbookAfterXMLExport**	XML-файл экспортирован
WorkbookAfterXMLImport**	XML-файл импортирован или XML-данные обновились из источника
WorkbookAddinUninstall	Удалена надстройка
WorkbookBeforeClose	Закрыта любая открытая рабочая книга
WorkbookBeforePrint	Напечатана любая рабочая книга
WorkbookBeforeSave	Сохранена любая рабочая книга
WorkbookBeforeXMLExport**	XML-файл готов к экспорту или источник XML-данных готов к обновлению
WorkbookBeforeXMLImport**	XML-файл готов к импорту
WorkbookDeactivate	Деактивизирована любая открытая рабочая книга
WorkbookNewSheet	Новый лист создан в любой открытой рабочей книге
WorkbookOpen	Открыта рабочая книга
WorkBookPivotTableCloseConnection*	Закрыто соединение с внешним источником данных сводной таблицы
WorkbookPivotTableOpenConnection*	Открыто соединение с внешним источником данных сводной таблицы
WorkbookSync**	Рабочая книга, часть рабочего пространства документа, синхронизируется с копией на сервере

* *Эти события поддерживаются только в Excel 2002 и выше*

** *Эти события поддерживаются только в профессиональном выпуске Excel 2003*

Включение событий уровня объекта Application

Для того чтобы использовать события уровня объекта `Application`, необходимо выполнить следующие действия.

1. Создайте новый модуль класса.
2. Присвойте имя этому модулю класса в окне Properties. По умолчанию VBA присваивает каждому новому модулю класса такие имена, как `Class1`, `Class2` и т.д. Рекомендуется использовать более описательное имя.
3. В модуле класса объявите глобальный объект типа `Application`, используя ключевое слово `WithEvents`.

   ```
   Public WithEvents XL As Application
   ```

4. Создайте переменную, которая будет использоваться в качестве ссылки на объект `Application` в модуле класса. Это должна быть переменная уровня модуля, объявленная в обычном модуле VBA (а не в модуле класса).

   ```
   Dim X As New clsApp
   ```

5. Свяжите объявленную переменную с объектом `Application`. Обычно эта операция выполняется в процедуре `Workbook_Open`.

   ```
   Set X.XL = Application
   ```

6. Создайте процедуры обработки событий и поместите их в модуль класса.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Эти действия практически идентичны выполняемым при обработке событий на уровне объекта встроенной диаграммы. Дополнительная информация по обработке событий встроенной диаграммы приводится в главе 18.

Определение факта открытия рабочей книги

Пример, приведенный в этом разделе, отслеживает события открытия каждой рабочей книги и сохранения информации в текстовом файле. Начать создание такой процедуры можно со вставки нового модуля класса, называющегося `AppClass`. В модуле класса должен располагаться следующий код.

```
Public WithEvents AppEvents As Application

Private Sub AppEvents_WorkbookOpen _
  (ByVal Wb As Excel.Workbook)
    Call UpdateLogFile(Wb)
End Sub
```

Этот код объявляет объект `Application`, который называется `AppEvents` и поддерживает обработку событий. Процедура `AppEvents_WorkbookOpen` вызывается каждый раз при открытии рабочей книги. Эта процедура обработки события вызывает процедуру `UpdateLogFile` и передает ей аргумент `Wb`, который представляет открытую рабочую книгу. После этого необходимо добавить модуль VBA, содержащий следующий код.

```
Sub UpdateLogFile(Wb)
    On Error Resume Next
    txt = Wb.FullName
    txt = txt & "," & Date & "," & Time
    txt = txt & "," & Application.UserName
```

```
     Fname = ThisWorkbook.Path & "\logfile.txt"
     Open Fname For Append As #1
     Write #1, txt
     Close #1
     MsgBox, txt
End Sub
```

Обратите внимание на объявление переменной типа `AppClass` (так называется модуль класса). Вызов процедуры `Init` производится в процедуре `Workbook_Open`, которая расположена в модуле кода объекта `ЭтаКнига`. Процедура `Workbook_Open` выглядит следующим образом.

```
Private Sub Workbook_Open()
    Call Init
End Sub
```

Процедура `UpdateLogFile` открывает текстовый файл (или создает его, если такого файла не существует) и записывает ключевую информацию об открытой рабочей книге: имя файла, полный путь к нему, дату, время и имя пользователя.

Процедура `Workbook_Open` вызывает процедуру `Init`. Таким образом, при открытии рабочей книги процедура `Init` создает новый объект.

Этот пример доступен на прилагаемом к книге компакт-диске. Удостоверьтесь, что файл скопирован на жесткий диск перед использованием. Текстовый файл записывается в ту папку, в которой находится рабочая книга.

Отслеживание событий уровня объекта Application

Для того чтобы "прочувствовать" процесс генерации событий, воспользуйтесь списком событий, которые генерируются в процессе повседневной работы.

На прилагаемом к книге компакт-диске содержится рабочая книга, отображающая каждое событие уровня объекта `Application`, которое происходит в процессе работы (рис. 19.7).

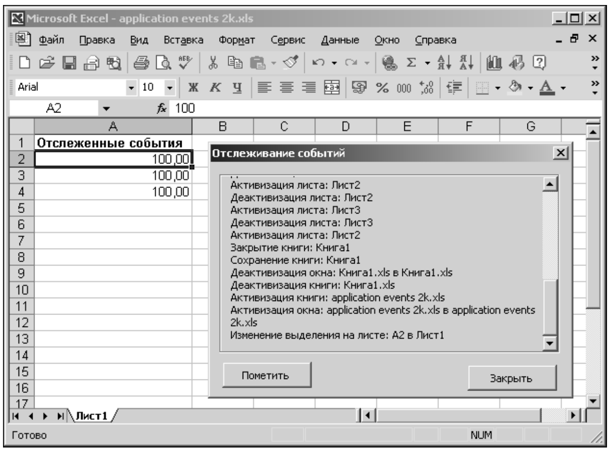


Рис. 19.7. Эта рабочая книга использует модуль класса для отслеживания всех событий уровня объекта `Application`

Рабочая книга содержит модуль с 21-ой процедурой (по одной на каждое событие, поддерживаемое объектом `Application`). Приведем пример такой процедуры.

```
Private Sub XL_NewWorkbook(ByVal Wb As Excel.Workbook)
    LogEvent "Создание книги: " & Wb.Name
End Sub
```

Каждая процедура вызывает процедуру `LogEvent` и передает ей в качестве аргумента имя события и объект. Ниже приведен код процедуры `LogEvent`.

```
Sub LogEvent(txt)
    EventNum = EventNum + 1
    With UserForm1
        With .lblEvents
            .AutoSize = False
            .Caption = .Caption & vbCrLf & txt
            .Width = UserForm1.FrameEvents.Width - 20
            .AutoSize = True
        End With
        .FrameEvents.ScrollHeight = .lblEvents.Height + 20
        .FrameEvents.ScrollTop = EventNum * 20
    End With
End Sub
```

Процедура `LogEvent` обновляет диалоговое окно `UserForm`, изменяя свойство `Caption` элемента управления `Label`, который называется `lblEvents`. Кроме того, процедура изменяет значения свойств `ScrollHeight` и `ScrollTop` элемента управления `Frame`, называющегося `FrameEvents`. Элемент управления `Label` расположен внутри элемента управления `Frame`.

События объекта UserForm

Объект `UserForm` поддерживает достаточно большое количество событий, а каждый элемент управления, размещенный в пользовательском диалоговом окне, поддерживает собственный набор событий. В табл. 19.5 перечислены события объекта `UserForm`, которые можно перехватывать с помощью VBA.

Таблица 19.5. События, поддерживаемые объектом UserForm

Событие	Действие, которое приводит к возникновению события
`Activate`	Активизация диалогового окна
`AddControl`	Добавление элемента управления на этапе выполнения
`BeforeDragOver`	Операция перетаскивания в процессе выполнения; указатель мыши расположен над диалоговым окном
`BeforeDropOrPaste`	Пользователь собирается вставить или отпустить данные (т.е. момент непосредственного отпускания кнопки мыши)
`Click`	Щелчок мышью в тот момент, когда указатель находится над диалоговым окном
`DblClick`	Двойной щелчок мышью, когда указатель находится над диалоговым окном
`Deactivate`	Деактивизация диалогового окна
`Error`	Элемент управления вызывает ошибку, но он не может сообщить информацию об ошибке вызывающей программе
`Initialize`	Загрузка в память диалогового окна
`KeyDown`	Нажатие клавиши

Событие	Действие, которое приводит к возникновению события
`KeyPress`	Нажатие пользователем любой клавиши, приводящей к вводу символа
`KeyUp`	Отпускание клавиши
`LayOut`	Изменение размера диалогового окна
`MouseDown`	Нажатие кнопки мыши
`MouseMove`	Перемещение указателя мыши
`MouseUp`	Отпускание кнопки мыши
`QueryClose`	Происходит перед тем, как диалоговое окно закрывается
`RemoveControl`	Элемент управления удаляется из диалогового окна на этапе выполнения
`Resize`	Диалоговое окно изменяет размеры
`Scroll`	Диалоговое окно прокручивается
`Terminate`	Диалоговое окно прекращает свою работу
`Zoom`	Диалоговое окно изменило свой масштаб

Ряд примеров из глав 13–15 демонстрирует обработку событий для пользовательских диалоговых окон и расположенных в них элементов управления.

События, не связанные с конкретными объектами

События, которые рассматривались ранее, были связаны с конкретными объектами (`Application`, `Workbook`, `Sheet` и т.д.). В этом разделе будут описаны дополнительные события, которые не связаны с определенными объектами: `OnTime` и `OnKey`. Доступ к ним осуществляется с помощью методов объекта `Application`.

В отличие от других событий, которые рассматриваются в этой главе, события `OnKey` и `OnTime` программируются с помощью кода, расположенного в модуле VBA общего назначения.

Событие OnTime

Событие `OnTime` происходит в определенное время суток. Следующий пример демонстрирует обработку событий Excel таким образом, что ровно в 15:00 отображается окно сообщения и раздается звуковой сигнал.

```
Sub SetAlarm()
    Application.OnTime TimeValue("15:00:00"), "DisplayAlarm"
End Sub

Sub DisplayAlarm()
    Beep
    MsgBox "Просыпайся. Время пить кофе!"
End Sub
```

В представленном примере процедура `SetAlarm` использует метод `OnTime` объекта `Application` для настройки события `OnTime`. Этот метод принимает два аргумента: время (в данном случае это 3:00 p.m. или 15:00) и имя процедуры, которая будет запущена в указанное время (в приводимом примере используется процедура `DisplayAlarm`). После выполнения процедуры `SetAlarm` процедура `DisplayAlarm` будет вызвана в 15:00, что приведет к отображению окна сообщения, показанного на рис. 19.8.

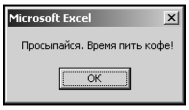


Рис. 19.8. Окно сообщения, отображаемое в определенное время суток

Если необходимо запланировать событие относительно текущего времени суток (например, через 20 минут), то воспользуйтесь следующим оператором.

```
Application.OnTime Now + TimeValue("00:20:00"), "DisplayAlarm"
```

Метод `OnTime` можно также использовать, если вы планируете вызов процедуры на определенную дату. Представленный далее оператор запустит процедуру `DisplayAlarm` 1 апреля 2004 года в 12:01.

```
Application.OnTime DateSerial(2004, 4, 1) + _
    TimeValue("00:00:01"), "DisplayAlarm"
```

НА ЗАМЕТКУ

Метод `OnTime` поддерживает два дополнительных аргумента. Если вы планируете использовать этот метод, то обратитесь к справочному руководству для получения более полной информации.

Приведенные ниже две процедуры демонстрируют методы управления повторяющимися событиями. В данном случае ячейка `A1` обновляется каждые пять секунд и в нее заносится текущее время. При выполнении процедур `UpdateClock` время сохраняется в ячейке `A1`, и эта операция повторяется через 5 секунд. Для прекращения события используется процедура `StopClock`. Обратите внимание, что `NextTick` — это переменная уровня модуля, которая сохраняется интервал времени перед наступлением следующего события.

```
Dim NextTick As Date
Sub UpdateClock()
ThisWorkbook.Sheets(1).Range("A1") = Time
NextTick = Now + TimeValue("00:00:01")
    Application.OnTime NextTick, "UpdateClock"
End Sub
```

ВНИМАНИЕ

Событие `OnTime` наступает даже тогда, когда рабочая книга закрывается. Другими словами, если вы закроете рабочую книгу без запуска процедуры `StopClock`, то первая повторно откроется через пять секунд (предполагается, что сама программа все еще запущена). Для предотвращения этого поведения рабочей книги используется процедура обработки событий `Workbook_BeforeClose` со следующим оператором.

```
Call StopClock
```

Событие OnKey

В процессе работы Excel постоянно отслеживает нажимаемые пользователем клавиши. Поэтому вы можете настроить клавишу или комбинацию клавиш, которые при нажатии запустят определенную процедуру.

В следующем примере используется метод `OnKey` для настройки события `OnKey`. В данном случае переназначается значение клавиш <PgUp> и <PgDn>. После выполнения процедуры `Setup_OnKey` нажатие клавиши <PgDn> приведет к запуску процедуры `PgDn_Sub`, а нажатие клавиши <PgUp> — процедуры `PgUp_Sub`. В итоге проведенных изменений нажатие клавиши <PgDn> вызывает перемещение курсора на одну строку вниз, а клавиши <PgUp> — на одну строку вверх.

```
Sub Setup_OnKey()
    Application.OnKey "{PgDn}", "PgDn_Sub"
    Application.OnKey "{PgUp}", "PgUp_Sub"
End Sub

Sub PgDn_Sub()
    On Error Resume Next
    ActiveCell.Offset(1, 0).Activate
End Sub

Sub PgUp_Sub()
    On Error Resume Next
    ActiveCell.Offset(-1, 0).Activate
End Sub
```

Обратите внимание на то, что коды клавиш заключены в фигурные скобки, а не в кавычки. Для получения полного списка кодов клавиатуры необходимо обратиться к справочному руководству. При поиске разделов укажите ключевое слово `OnKey`.

В предыдущих примерах использовался оператор `On Error Resume Next`, который позволял игнорировать все возникающие ошибки. Например, если активная ячейка находится в первой строке, то перемещение на одну строку вверх приводит к возникновению ошибки. Кроме того, если активным является лист диаграммы, то возникнет ошибка, так как на листе диаграммы не существует такого понятия, как активная ячейка.

Запустив следующую процедуру, можно отменить событие `OnKey` и вернуть клавишам их обычную функциональность.

```
Sub Cancel_OnKey()
    Application.OnKey "{PgDn}"
    Application.OnKey "{PgUp}"
End Sub
```

Использование пустой строки в качестве второго аргумента метода `OnKey` *не* приводит к отмене текущего события `OnKey`. Вместо этого Excel игнорирует нажатие указанных клавиш и не выполняет вообще никаких действий при их нажатии. Например, приведенный ниже оператор сообщает Excel, что необходимо игнорировать нажатие комбинации клавиш <Alt+F4> (символ процента представляет клавишу <Alt>).

```
Application.OnKey "%{F4}", ""
```

Несмотря на то, что событие `OnKey` допускается использовать с целью назначения определенной комбинации клавиш для запуска макроса, вы также можете воспользоваться диалоговым окном Параметры макроса. Дополнительная информация по этому вопросу представлена в главе 9.

Глава 20

Взаимодействие с другими приложениями

В ЭТОЙ ГЛАВЕ...

В данной главе будут рассмотрены способы, с помощью которых приложения Excel могут взаимодействовать с другими приложениями. Кроме того, вы будете иметь возможность ознакомиться с полезными примерами.

- Запуск или активизация другого приложения из Excel
- Отображение диалогового окна Windows Панель управления
- Использование средства автоматизации для управления другим приложением
- Простой пример использования ADO для получения данных
- Применение функции `SendKeys` в качестве последней возможности

На самых ранних этапах использования персональных компьютеров взаимодействие между приложениями практиковалось очень редко. До появления операционных систем с поддержкой многозадачности взаимодействие между приложениями ограничивалось импортом файлов. Даже копирование информации и вставка ее в другое приложение (то, что теперь доступно любому пользователю) ранее считалось невозможным.

В настоящее время большая часть программного обеспечения разрабатывается с поддержкой минимальных средств взаимодействия приложений. Многие приложения Windows поддерживают буфер обмена, используемый в операциях копирования и вставки данных между приложениями. Большинство программных продуктов для Windows поддерживают технологию DDE (Dynamic Data Exchange — Динамический обмен данными), а более новые продукты — и автоматизацию.

Запуск другого приложения

Иногда необходимо запустить другое приложение из Excel. Например, можно запустить программу установки соединения или даже командный файл DOS. Разработчик приложения также в состоянии облегчить пользователю доступ к папке Панель управления.

Использование функции Shell

Функция `Shell` упрощает процесс запуска других приложений. В листинге 20.1 представлена процедура, которая запускает приложение Windows Character Map (Таблица символов), позволяющее пользователю ввести специальный символ.

Листинг 20.1. Запуск приложения Windows

```
Sub RunCharMap()
    On Error Resume Next
    Program = "Charmap.exe"
    TaskID = Shell(Program, 1)
    If Err <> 0 Then
        MsgBox "Нельзя запустить " & Program, vbCritical, "Ошибка"
    End If
End Sub
```

Приложение, которое запускает эта процедура, показано на рис. 20.1.



Если используется Excel 2002 или выше, то программа Windows Character Map (Таблица символов) вам больше не потребуется, так как она дублируется командой Excel Вставка⇨Символ.

Функция `Shell` возвращает номер идентификатора приложения. Этот номер можно использовать позднее для активизации задачи. Второй аргумент функции `Shell` определяет способ отображения приложения (для окна нормального размера с последующей активизацией используется аргумент 1).

Если вызов функции `Shell` завершился неудачно, то будет создано сообщение об ошибке. Таким образом, эта процедура использует оператор `On Error` для отображения сообщения, если файл нельзя найти или произошла другая ошибка.

Важно понимать, что код VBA продолжает свою работу после запуска приложения, загруженного с помощью функции `Shell`. Другими словами, функция `Shell` запускает приложения *асинхронно*. Если в процедуре после вызова функции `Shell` находятся другие инструкции, то они выполняются одновременно с запущенной программой. Если инструкции требуют вмешательства пользователя (например, при отображении окна сообщения), то строка заголовка Excel будет мигать, пока активно другое приложение.

В некоторых случаях может понадобится запустить приложение с помощью функции `Shell`, но при этом код VBA должен прекратить выполняться, пока запущенное приложение не создаст файл, который будет использован далее в коде VBA. Несмотря на то, что выполнение кода приостановить невозможно, вы вправе создать цикл, который, кроме отслеживания состояния запущенного приложения, не выполняет никаких функций. В листинге 20.2 приведен пример окна сообщения, которое отображается на экране тогда, когда приложение, запущенное с помощью функции `Shell`, завершает свою работу.

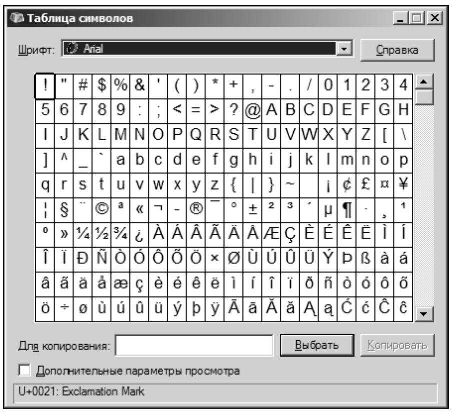


Рис. 20.1. Запуск программы Windows Character Map (Таблица символов) в Excel

Листинг 20.2. Ожидание завершения работы приложения

```
Declare Function OpenProcess Lib "kernel32" _
    (ByVal dwDesiredAccess As Long, _
    ByVal bInheritHandle As Long, _
    ByVal dwProcessId As Long) As Long

Declare Function GetExitCodeProcess Lib "kernel32" _
    (ByVal hProcess As Long, _
    lpExitCode As Long) As Long

Sub RunCharMap2()
    Dim TaskID As Long
    Dim hProc As Long
    Dim lExitCode As Long

    ACCESS_TYPE = &H400
    STILL_ACTIVE = &H103

    Program = "Charmap.exe"
    On Error Resume Next

'   Определение оболочки
    TaskID = Shell(Program, 1)

'   Обработка задачи
    hProc = OpenProcess(ACCESS_TYPE, False, TaskID)

    If Err <> 0 Then
        MsgBox "Нельзя запустить " & Program, vbCritical, "Ошибка"
        Exit Sub
    End If

    Do  'Непрерывный цикл
'       Проверка процесса
        GetExitCodeProcess hProc, lExitCode
 '       Разрешить обработку события
        DoEvents
    Loop While lExitCode = STILL_ACTIVE

'   Задача завершена, отображение сообщения
    MsgBox Program & " завершила свою работу"
End Sub
```

После запуска приложения данная процедура будет постоянно вызывать функцию `GetExitCodeProcess` в конструкции `Do Loop`. В цикле проверяется значение переменной `lExitCode`. Когда программа завершает свою работу, `lExitCode` изменяет свое значение, цикл завершается, и код VBA продолжает выполняться.

НА КОМПАКТ-ДИСКЕ

Оба примера доступны на прилагаемом к книге компакт-диске.

Использование API-функции ShellExecute

`ShellExecute` — это функция Windows API, позволяющая запускать другие приложения. Примечательно, что другое приложение запускается только в том случае, когда открываемый файл имеет один из зарегистрированных в системе типов. Например, вы можете воспользоваться функцией `ShellExecute` для открытия Web-документа в окне браузера.

Ниже представлено объявление этой функции API (данный код необходимо разместить в начале модуля VBA).

```
Private Declare Function ShellExecute Lib "shell32.dll" _
  Alias "ShellExecuteA" (ByVal hWnd As Long, _
  ByVal lpOperation As String, ByVal lpFile As String, _
  ByVal lpParameters As String, ByVal lpDirectory As String, _
  ByVal nShowCmd As Long) As Long
```

Следующая процедура демонстрирует вызов функции `ShellExecute`. В данном случае с ее помощью запускается графический редактор, посредством которого в системе просматриваются GIF-файлы.

```
Sub ShowGraphic()
    Dim FileName As String
    FileName = "c:\face.gif"
    Call ShellExecute(0&, vbNullString, FileName, _
        vbNullString, vbNullString, vbNormalFocus)
End Sub
```

Следующая процедура демонстрирует вызов браузера.

```
Sub OpenFile()
    Dim URL As String

    URL = "http://www.microsoft.com"
    Call ShellExecute(0&, vbNullString, URL, _
        vbNullString, vbNullString, vbNormalFocus)
End Sub
```

Эта же методика применяется и по отношению к почтовым сообщениям. В приведенном ниже примере создается сообщение по умолчанию, в котором указан адрес получателя.

```
Sub StartEmail()
    Dim Addr As String
    Addr = "mailto:bgates@microsoft.com"
    Call ShellExecute(0&, vbNullString, Addr, _
        vbNullString, vbNullString, vbNormalFocus)
End Sub
```

Активизация другого приложения

Остерегайтесь потенциальной проблемы: может оказаться, что приложение уже работает в момент использования функции `Shell` для его запуска. Это приведет к запуску нового сеанса указанного приложения. Вам же требуется перейти к уже запущенному сеансу работы программы, а не создавать новый.

Оператор AppActivate

Представленная далее процедура `StartCalculator` использует оператор `AppActivate` для активизации приложения, если оно уже запущено (в данном случае активизируется программа Microsoft Calculator (Калькулятор)). В качестве аргумента оператора `AppActivate` используется название приложения. Если оператор `AppActivate` возвращает ошибку, то приложение Calculator (Калькулятор) еще не запущено. Далее процедура запускает указанное приложение.

```
Sub StartCalculator()
    AppFile = "Calc.exe"
    On Error Resume Next
```

```
    AppActivate "Calculator"
    If Err <> 0 Then
        Err = 0
        CalcTaskID = Shell(AppFile, 1)
        If Err <> 0 Then MsgBox "Невозможно запустить"
    End If
End Sub
```

НА КОМПАКТ-ДИСКЕ

Этот пример доступен на прилагаемом к книге компакт-диске.

Активизация приложения Microsoft Office

Если запускаемое приложение входит в состав пакета Microsoft Office, то вы можете использовать метод `ActivateMicrosoftApp` объекта `Application`. Например, следующая процедура запускает программу Word.

```
Sub StartWord()
     Application.ActivateMicrosoftApp xlMicrosoftWord
End Sub
```

Если Word уже запущена, то предыдущая процедура просто активизирует его. В используемом методе можно применять и другие константы.

- `xlMicrosoftPowerPoint`
- `xlMicrosoftMail`
- `xlMicrosoftAccess`
- `xlMicrosoftFoxPro`
- `xlMicrosoftProject`
- `xlMicrosoftSchedulePlus`

Запуск аплетов папки Панель управления и мастеров

Windows предоставляет в распоряжение пользователя большое количество системных диалоговых окон и мастеров, многие из которых расположены в окне Панель управления. В приложении Excel может понадобится отобразить одно или несколько диалоговых окон из этой папки. Например, вы имеете возможность отобразить диалоговое окно Свойства: Дата и время, которое показано на рис. 20.2.

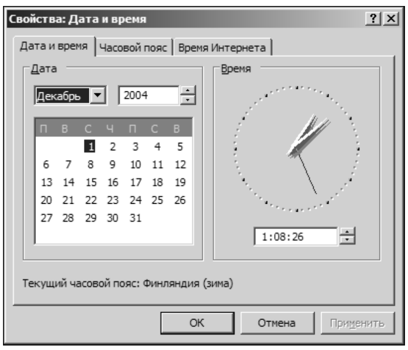


Рис. 20.2. Использование кода VBA для отображения аплета папки Панель управления

Основным условием для запуска системных диалоговых окон является указание правильного значения аргумента функции `Shell`.

Следующая процедура использует значение аргумента, необходимое для открытия диалогового окна Свойства: Дата и время.

```
Sub ShowDateTimeDlg()
  Arg = "rundll32.exe shell32.dll,Control_RunDLL timedate.cpl"
  On Error Resume Next
  TaskID = Shell(Arg)
  If Err <> 0 Then
      MsgBox ("Невозможно запустить приложение")
  End If
End Sub
```

Ниже приведен общий формат приложения `rundll32.exe`.

```
rundll32.exe shell32.dll,Control_RunDLL имя_файла.cpl,@n,t
```

- `имя_файла.cpl`. Имя одного из файлов CPL папки Панель управления.
- `n`. Номер аплета в файле CPL.
- `t`. Количество вкладок (для многовкладочных окон).

Рабочая книга, демонстрирующая использование 12 значений этого аргумента, показана на рис. 20.3 (она содержится на прилагаемом к книге компакт-диске).

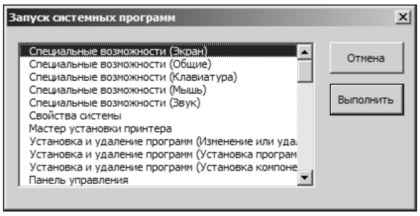


Рис. 20.3. Эта форма позволяет отображать системные диалоговые окна в Excel

Автоматизация

Вы можете создать макрос Excel, который будет управлять программой Microsoft Word. Точнее, макрос Excel будет контролировать самый важный компонент Word — так называемый *сервер автоматизации*. В подобных случаях Excel выступает клиентским, а Word — серверным приложением. Более того, можно создать приложение Visual Basic, которое управляет работой Excel. Процесс управления одного приложения другим часто называют *OLE-автоматизацией*, или *ActiveX-автоматизацией* (известно, что Microsoft часто меняет ею же разработанную терминологию).

Под автоматизацией подразумевается достаточно привлекательная технология. Разработчик, которому необходимо создать диаграмму, может обратиться к другому приложению, получить от него объект `Chart` и использовать свойства и методы этого объекта. Автоматизация, в некотором смысле, размывает границы между приложениями. Конечный пользователь может работать с объектом Access и даже об этом не подозревать.

Некоторые приложения (и Excel в том числе) могут выступать как клиентскими, так и серверными программами. Отдельные приложения функционируют только как клиентские или только как серверные программы.

В этом разделе продемонстрированы способы использования VBA для получения доступа и для управления объектами, которые предоставляются другими приложениями. В приведенных примерах используется только программа Microsoft Word, но рассматриваемая концепция в одинаковой степени относится ко всем приложениям, которые предоставляют собственные объекты для автоматизации (с течением времени таких приложений становится все больше).

Работа с внешними объектами

Как известно, в Excel можно использовать команду Вставка⇨Объект для встраивания объекта (например, документа Word) в рабочий лист. Кроме того, существует возможность создать объект и манипулировать им в VBA. (Это возможно благодаря все той же технологии автоматизации.) При таком подходе программе предоставляется полный доступ к объекту. Разработчикам подобная технология обычно нравится больше, чем встраивание объекта в рабочий лист. Когда объект встроен, пользователь должен точно знать, как пользоваться приложением, которому принадлежит объект. Но если для управления объектом применяется код VBA, можно настроить объект так, что пользователь будет управлять им с помощью таких простых операций, как щелчок на кнопке.

Ранняя и поздняя привязка

Перед началом работы с внешним объектом необходимо создать его экземпляр. Данного результата можно достичь двумя способами: с помощью ранней или поздней привязки. *Привязка* обозначает установку соответствия между вызовами тех функций, которые создает разработчик, и фактическим кодом, выполняемым при запуске функции.

РАННЯЯ ПРИВЯЗКА

На этапе разработки можно создать ссылку на библиотеку объектов с помощью команды Tools⇨References редактора VBE. Выполнение этой команды приведет к отображению диалогового окна, показанного на рис. 20.4.

После создания ссылки на библиотеку объектов можно использовать окно Object Browser (показанное на рис. 20.5) для просмотра имен, методов и свойств объектов.

При использовании ранней привязки необходимо создать ссылку на конкретную версию библиотеки объектов. Например, можно указать Microsoft Word 8.0 Object Library, Microsoft Word 9.0 Object Library, Microsoft Word 10.0 Object Library или Microsoft Word 11.0 Object Library. После этого используется следующий оператор для создания объекта.

```
Dim WordApp As New Word.Application
```

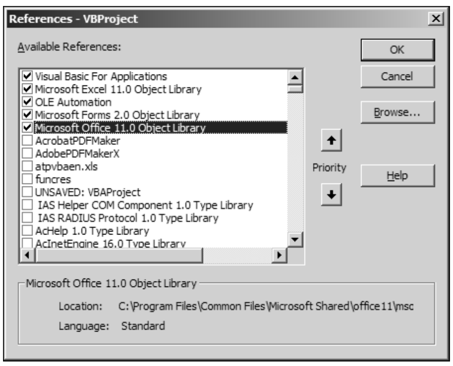


Рис. 20.4. Добавление ссылки на файл библиотеки объектов

Применение ранней привязки для создания объекта с помощью ссылки на библиотеку объектов является более эффективным методом, который обеспечивает большую производительность приложения. Ранняя привязка может использоваться только тогда, когда контролируемый объект имеет отдельный файл библиотеки типов или библиотеки объектов. Кроме того, следует удостовериться, что в компьютере пользователя установлена необходимая версия библиотеки объектов.

Ранняя привязка позволяет применять константы, которые объявлены в библиотеке объектов. Например, Word (как и Excel) содержит ряд предопределенных констант, которые можно использовать в коде VBA. В ранней привязке в создаваемом коде можно вводить эти константы. Если используется поздняя привязка, вам придется обратиться к фактическому значению константы, а не к ее имени. Еще одним преимуществом ранней привязки является возможность использования окна Object Browser и параметра Auto List Members в редакторе VBE для получения простого способа доступа к свойствам и методам объектов. Такая возможность не представлена в поздней привязке, так как тип объекта определяется только на этапе выполнения.

ПОЗДНЯЯ ПРИВЯЗКА

На этапе выполнения можно использовать или функцию `CreateObject` для создания объекта, или функцию `GetObject` для получения сохраненного объекта. Такие объекты объявляются как объекты универсального типа `Object`. Ссылка на объекты задается на этапе выполнения.

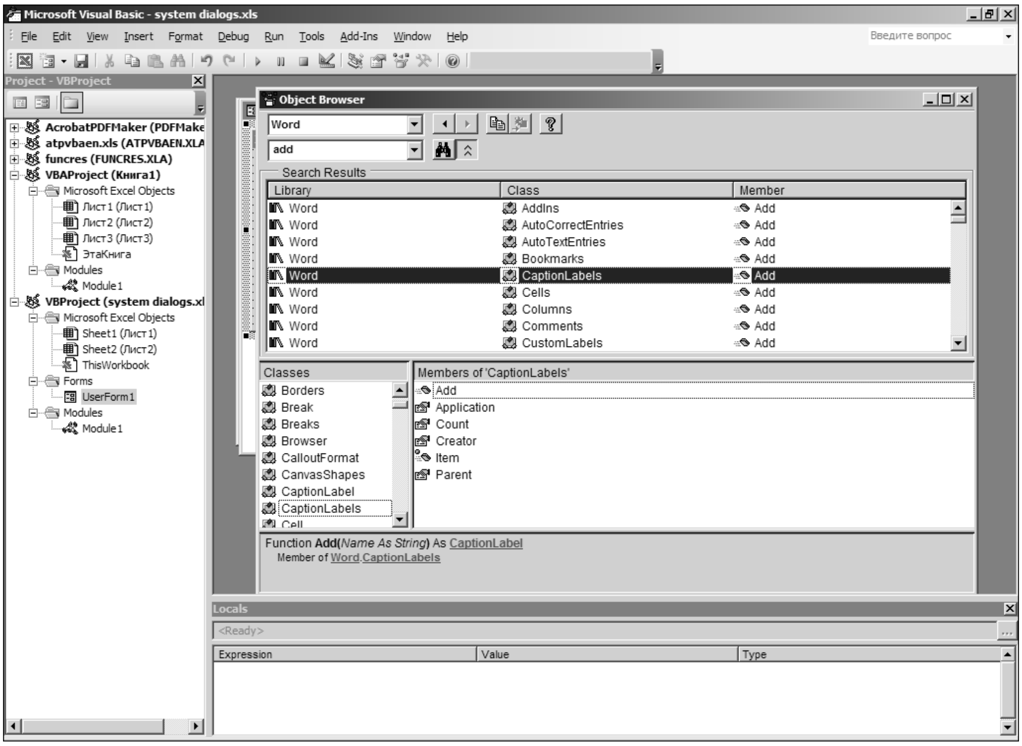


Рис. 20.5. Использование окна Object Browser *для получения информации об объектах в библиотеке*

Позднюю привязку можно применить даже тогда, когда неизвестна версия библиотеки объектов, установленной в операционной системе. Например, следующий код, который управляет Word 97 и более поздними версиями, создает объект Word.

```
Dim WordApp As Object
Set WordApp = CreateObject("Word.Application")
```

Если в системе установлено несколько версий Word, то можно создать объект определенной версии. Приведенный ниже оператор управляет объектом Word 2000.

```
Set WordApp = CreateObject("Word.Application.9")
```

Параметр в реестре Windows для объекта Automation программы Word, а также ссылка на объект `Application` в VBA оказались одинаковыми: `Word.Application`. Но они указывают на разные элементы. Когда объект объявляется с помощью оператора `As Word.Application` или `As New Word.Application`, то этот термин обозначает объект `Application` в библиотеке `Word`. А если вызвать функцию `CreateObject("Word.Application")`, то термин будет ссылаться на параметр, под которым последняя версия Word известна в системном реестре Windows. Данное утверждение не относится ко всем объектам автоматизации, хотя оно и справедливо для основных компонентов пакета Office. Если пользователь заменит Word 2000 на Word 2003, то функция `CreateObject("Word.Application")` будет действительной — она ссылается на новое приложение. Но если удалить Word 2003, то функция `CreateObject("Word.Application.11")`, обращающаяся непосредственно к Word 2003, работать не будет.

Функция `CreateObject`, которая обращается к таким объектам автоматизации, как `Word.Application` или `Excel.Application`, всегда возвращает новый экземпляр объекта, т.е. запускается новая копия приложения. Поэтому в случае, когда объект автоматизации уже запущен, запускается новый экземпляр, после чего создается новый объект указанного типа.

Для того чтобы использовать текущий экземпляр или чтобы запустить приложение и заставить его загрузить сохраненный файл, воспользуйтесь функцией `GetObject`.

Если необходимо использовать автоматизацию в приложениях Office, то рекомендуется выполнить раннюю привязку к самой ранней версии программного продукта, который установлен в системе. Например, если требуется реализовать автоматизацию с Word 97, Word 2000 и Word 2002, то следует использовать библиотеку объектов Word 97, чтобы обеспечить совместимость со всеми тремя версиями. Таким образом гарантируется невозможность использования уникальных функций, предоставляемых более поздними версиями Word.

Функции `GetObject` и `CreateObject`

Функции VBA `GetObject` и `CreateObject` возвращают ссылки на объект, но работают совершенно по-разному.

Функция `CreateObject` используется для создания интерфейса нового экземпляра приложения. Эту функцию необходимо использовать, если приложение еще не запущено. Если экземпляр приложения уже существует, то будет создан новый экземпляр. Например, представленный далее оператор загружает Excel и возвращает в переменную `XLApp` ссылку на созданный объект `Excel.Application`.

```
Set XLApp = CreateObject("Excel.Application")
```

Функция `GetObject` используется приложением, которое уже запущено, или применяется для загрузки файла в запускаемом приложении. Следующий оператор, например, запускает Excel и загружает в него файл `Myfile.xls`. В переменную `XLBook` возвращается ссылка на объект `Workbook` (файл `Myfile.xls`).

```
Set XLBook = GetObject("C:\Myfile.xls")
```

Простой пример поздней привязки

Приведенный пример демонстрирует создание объекта `Word` с помощью поздней привязки. Рассматриваемая процедура создает объект, отображает номер версии, закрывает приложение Word и уничтожает созданный объект (освобождая используемую память).

```
Sub GetWordVersion()
    Dim WordApp As Object
    Set WordApp = CreateObject("Word.Application")
    MsgBox WordApp.Version
    WordApp.Quit
    Set WordApp = Nothing
End Sub
```

Созданный объект `Word` остается невидимым. Если необходимо, чтобы в процессе обработки объект `Word` отображался на экране, то установите его свойство `Visible` в значение `True` с помощью следующего оператора.

```
WordApp.Visible = True
```

Данный пример можно запрограммировать с помощью ранней привязки. Однако вначале воспользуйтесь командой Tools⇨References для создания ссылки на библиотеку объектов Word. После этого можно применить приведенный ниже код.

```
Sub GetWordVersion()
    Dim WordApp As New Word.Application
    MsgBox WordApp.Version
    WordApp.Quit
    Set WordApp = Nothing
End Sub
```

Управление приложением Word из Excel

Пример, представленный в этом разделе, отображает сеанс автоматизации программы Word в Excel. Процедура `MakeMemos` создает три заметки в Word, после чего сохраняет их в отдельных файлах. Информация, которая использовалась для создания заметок, показана на рис. 20.6.

Процедура `MakeMemos`, отображенная в листинге 20.3, начинается с запуска объекта `WordApp`. После этого просматриваются три строки данных на листе `Лист1`, затем используются свойства и методы объекта `Word` для создания каждой заметки и сохранения ее в отдельном файле. Диапазон `Message` (в ячейке `E6`) содержит текст, который применяется для создания записок.

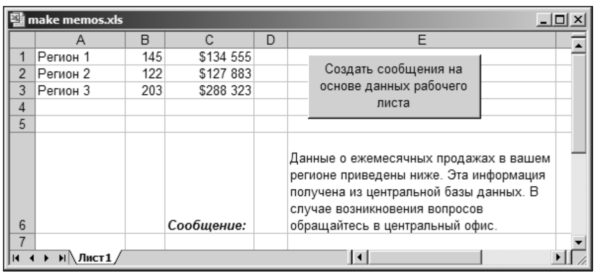


Рис. 20.6. Word автоматически генерирует три заметки на основе данных приложения Excel

Листинг 20.3. Генерация документов Word на основе данных в Excel

```
Sub MakeMemos()
'   Создание заметок в Word с помощью автоматизации (поздняя привязка)
    Dim WordApp As Object
    Dim Data As Range, message As String
    Dim Records As Integer, i As Integer
    Dim Region As String, SalesAmt As String, SalesNum As String
    Dim SaveAsName As String

'   Запуск Word и создание объекта
    Set WordApp = CreateObject("Word.Application")

'   Информация с рабочего листа

    Set Data = Sheets("Лист1").Range("A1")
    message = Sheets("Лист1").Range("Message")

'   Просмотр записей в Лист1
    Records = Application.CountA(Sheets("Лист1").Range("A:A"))
    For i = 1 To Records
'       Обновление сообщения в строке состояния
        Application.StatusBar = "Создание сообщений " & i

'       Назначение данных переменным
        Region = Data.Cells(i, 1).Value
        SalesNum = Data.Cells(i, 2).Value
        SalesAmt = Format(Data.Cells(i, 3).Value, "#,000")


'       Определение имен файлов
        SaveAsName = ThisWorkbook.Path & "\" & Region & ".doc"

'       Передача команд в Word
        With WordApp
            .Documents.Add
            With .Selection
                .Font.Size = 14
                .Font.Bold = True
                .ParagraphFormat.Alignment = 1
                .TypeText Text:="М Е М О Р А Н Д У М"
                .TypeParagraph
                .TypeParagraph
                .Font.Size = 12
                .ParagraphFormat.Alignment = 0
                .Font.Bold = False
                .TypeText Text:="Дата:" & vbTab & _
                    Format(Date, "mmmm d, yyyy")
                .TypeParagraph
                .TypeText Text:="Кому: менеджеру " & vbTab & Region
                .TypeParagraph
                .TypeText Text:="От:" & vbTab & _
                   Application.UserName
                .TypeParagraph
                .TypeParagraph
                .TypeText message
                .TypeParagraph
                .TypeParagraph
                .TypeText Text:="Продано товара:" & vbTab & SalesNum
                .TypeParagraph
                .TypeText Text:="На сумму:" & vbTab & _
                  Format(SalesAmt, "$#,##0")
            End With
                .ActiveDocument.SaveAs FileName:=SaveAsName
        End With
    Next i
```

```
'   Удаление объекта
    WordApp.Quit
    Set WordApp = Nothing

'   Обновление строки состояния
    Application.StatusBar = ""
    MsgBox Records & " заметки создано и сохранено в папке " & ThisWorkbook.Path
End Sub
```

На рис. 20.7 показан документ, созданный с помощью процедуры `MakeMemos`.

Создание этого макроса состояло из нескольких этапов. Все началось с записи макроса в Word. Записывались действия по созданию нового документа, добавлению и форматированию текста, а также сохранению файла. Этот макрос Word предоставил необходимую информацию о свойствах и методах объекта `Word`. После этого макрос был скопирован в модуль кода Excel. Обратите внимание, что использовался оператор `With-End With`. Перед каждой инструкцией между `With` и `End With` была добавлена точка. Например, первоначальный макрос Word содержал (среди прочих) приведенный ниже оператор.

```
Documents.Add
```

Макрос был модифицирован следующим образом.

```
With WordApp
    .Documents.Add
'   другие операторы
End With
```

Макрос, записанный в Word, использовал несколько встроенных констант. По причине того, что в данном примере используется поздняя привязка, пришлось заменить константы их фактическими значениями. Значения можно узнать в окне Immediate редактора VBE в Word.

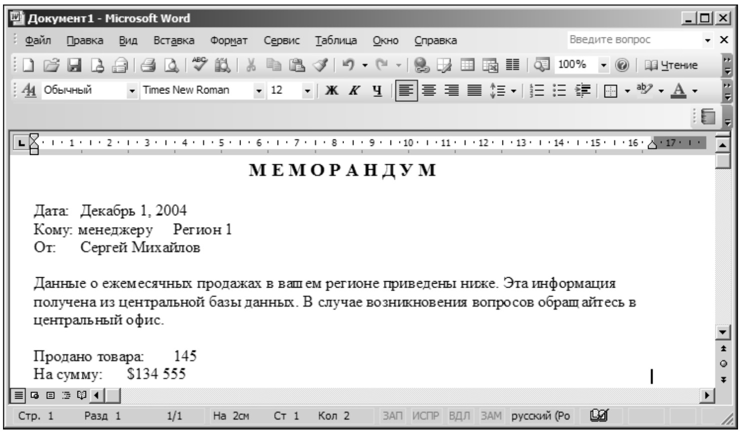


Рис. 20.7. Этот документ создан с помощью макроса Excel

Управление Excel из другого приложения

Вы имеете возможность управлять Excel из другого приложения (из программы Visual Basic или из макроса Word) — например, если необходимо провести некоторые расчеты в Excel, а результаты передать обратно в документ Word.

Создайте один из следующих объектов Excel с помощью указанной рядом функции.

- Объект `Application` — функция `CreateObject("Excel.Application")`.
- Объект `Workbook` — функция `CreateObject("Excel.Sheet")`.
- Объект `Chart` — функция `CreateObject("Excel.Chart")`.

В листинге 20.4 показана процедура, которая расположена в модуле кода VBA документа Word 2003. Эта процедура создает объект Excel `Worksheet` (представленный параметром `Excel.Sheet`) на основе существующей рабочей книги.

Листинг 20.4. Создание рабочего листа Excel из документа Word

```
Sub MakeExcelChart()
    Dim XLSheet As Object

'   Создание документа
    Documents.Add

'   Запрос значений
    StartVal = InputBox("Начальное значение")
    PctChange = InputBox("Относительное изменение")

'   Создание объекта Sheet
    Wbook = ThisDocument.Path & "\projections.xls"
    Set XLSheet = GetObject(Wbook, "Excel.Sheet").ActiveSheet

'   Добавление значений на лист
    XLSheet.Range("StartingValue") = StartVal
    XLSheet.Range("PctChange") = PctChange
    XLSheet.Calculate

'   Вставка заголовка страницы
    Selection.Font.Size = 14
    Selection.Font.Bold = True
    Selection.TypeText "Месячный прирост: " & _
      Format(PctChange, "0.0%")
    Selection.TypeParagraph
    Selection.TypeParagraph

'   Копирование данных листа и вставка в документ
    XLSheet.Range("data").Copy
    Selection.Paste

'   Копирование диаграммы и вставка в документ
    XLSheet.ChartObjects(1).Copy
    Selection.PasteSpecial _
        Link:=False, _
        DataType:=wdPasteMetafilePicture, _
        Placement:=wdInLine, DisplayAsIcon:=False

'   Удаление объекта
    Set XLSheet = Nothing
End Sub
```

Первоначальная рабочая книга показана на рис. 20.8. Процедура `MakeExcelChart` запрашивает у пользователя два значения и вставляет их в рабочий лист.

Пересчет листа приводит к обновлению диаграммы. После этого данные и диаграмма копируются из объекта Excel и вставляются в новый документ. Результат показан на рис. 20.9.

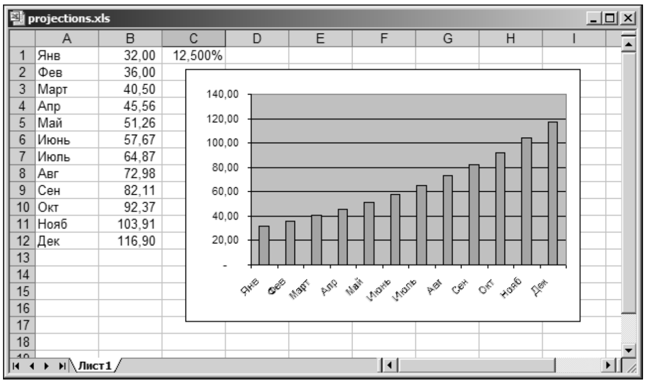


Рис. 20.8. Процедура VBA в Word использует этот рабочий лист

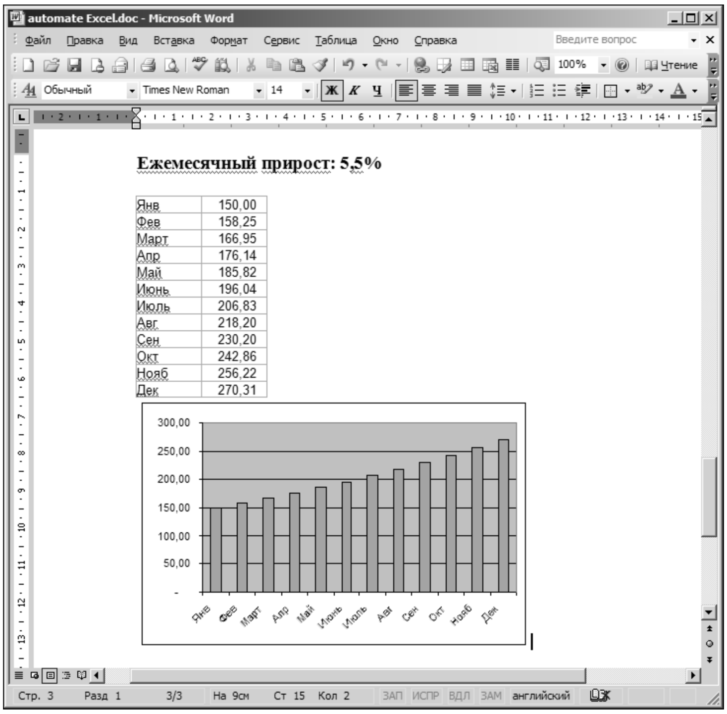


Рис. 20.9. Процедура VBA в Word использует Excel для создания документа

Отправка почтовых сообщений с помощью Outlook

Пример этого раздела иллюстрирует методы автоматизации с программой Microsoft Outlook.

На рис. 20.10 показан рабочий лист, который содержит данные почтового сообщения: имя, почтовый адрес и сумму предоставляемого пособия. Процедура листинга 20.5 просматривает строки (записи) этой рабочей книги и создает на основе полученных данных отдельные сообщения (Сохраненные в переменной `Msg`).

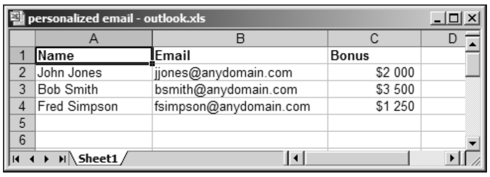


Рис. 20.10. Информация, используемая почтовым клиентом для создания новых сообщений

Листинг 20.5. Отправка почтовых сообщений из Excel

```
Sub SendEmail()
  'Используется ранняя привязка
  'Требует ссылки на библиотеку объектов Outlook
  Dim OutlookApp As Outlook.Application
  Dim MItem As Outlook.MailItem
  Dim cell As Range
  Dim Subj As String
  Dim EmailAddr As String
  Dim Recipient As String
  Dim Bonus As String
  Dim Msg As String

  'Создание объекта Outlook
  Set OutlookApp = New Outlook.Application

  'Просмотр данных столбцов
  For Each cell In Columns("B").Cells.SpecialCells(xlCellTypeConstants)
    If cell.Value Like "*@*" Then
      'Получение данных
      Subj = "Ваша премия"
      Recipient = cell.Offset(0, -1).Value
      EmailAddr = cell.Value
      Bonus = Format(cell.Offset(0, 1).Value, "$0,000.")

     'Составление сообщения
      Msg = "Дорогой  " & Recipient & vbCrLf & vbCrLf
      Msg = Msg & "Рад известить вас о начислении вам премии "
      Msg = Msg & Bonus & vbCrLf & vbCrLf
      Msg = Msg & "Вилли Билли" & vbCrLf
      Msg = Msg & "Президент"

      'Создание элемента сообщения и его отправка
      Set MItem = OutlookApp.CreateItem(olMailItem)
      With MItem
        .To = EmailAddr
        .Subject = Subj
        .Body = Msg
        .Send
      End With
    End If
  Next
End Sub
```

Этот пример демонстрирует раннее связывание, поэтому в нем применяются ссылки на библиотеку объектов Outlook. Обратите внимание, что используется два объекта: `Outlook` и `MailItem`. Первый был создан с помощью следующего оператора.

```
Set OutlookApp = New Outlook.Application
```

Объект `MailItem` создается таким оператором.

```
Set Mitem = OutlookApp.CreateItem(olMailItem)
```

Этот пример доступен на прилагаемом к книге компакт-диске.

В следующих разделах этой главы мы еще вернемся к теме отправки сообщений с помощью Excel.

Работа с ADO

ADO (ActiveX Data Object) является объектной моделью, которая позволяет осуществлять доступ к данным, хранящимся в различных форматах. Важно то, что такая методика позволяет использовать единую объектную модель для всех баз данных объектов. Представляемая технология доступа к данным является самой предпочтительной. Однако не путайте ее с DAO (Data Access Objects).

В настоящем разделе приводится простой пример использования ADO для получения данных из базы данных Access.

Программирование с использованием ADO — это сложная и серьезная тема. Если в приложении Excel вы решили получить доступ к внешним данным, то вам потребуются более глубокие знания. Обратитесь к книгам по программированию, которые рассматривают этот вопрос подробнее.

В следующем примере данные будут получены из базы данных Access, которая называется `budget.mdb`. Эта база данных содержит одну таблицу (Budget). Таблица состоит из семи полей. В рассматриваемом примере запрашиваются только записи, в которых поле `Пункт` содержит значение *Рента*, а поле `Подразделение` — значение *С. Америка*. Полученные данные хранятся в объекте `Recordset`. После получения данные передаются на рабочий лист (рис. 20.11).

```
Sub ADO_Demo()
'   Этот пример требует использования ссылки на
'   библиотеку Microsoft ActiveX Data Objects 2.x

    Dim DBFullName As String
    Dim Cnct As String, Src As String
    Dim Connection As ADODB.Connection
    Dim Recordset As ADODB.Recordset
    Dim Col As Integer

    Cells.Clear
    MsgBox "Это приложение извлекает данные из записей _
            Пункт = Рента и Подразделение = С. Америка"

'   Информация о базе данных
    DBFullName = ThisWorkbook.Path & "\budget.mdb"
```

```
'   Открытие соединения
    Set Connection = New ADODB.Connection
    Cnct = "Provider=Microsoft.Jet.OLEDB.4.0; "
    Cnct = Cnct & "Data Source=" & DBFullName & ";"
    Connection.Open ConnectionString:=Cnct

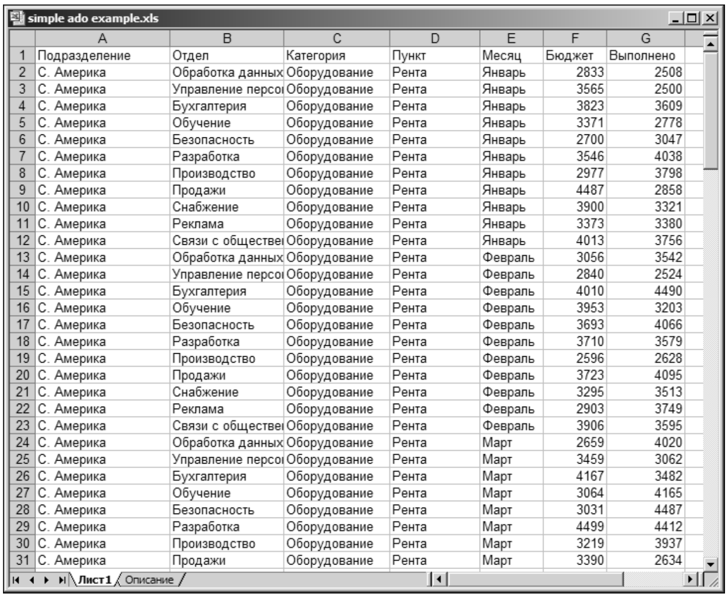
'   Создание объекта RecordSet
    Set Recordset = New ADODB.Recordset
    With Recordset
'       Фильтр
        Src = "SELECT * FROM Budget WHERE Пункт = 'Рента' "
        Src = Src & "and Подразделение = 'С. Америка'"
        .Open Source:=Src, ActiveConnection:=Connection

'       Запись имен полей
        For Col = 0 To Recordset.Fields.Count - 1
           Range("A1").Offset(0, Col).Value = Recordset.Fields(Col).Name
        Next

'       Запись данных
        Range("A1").Offset(1, 0).CopyFromRecordset Recordset
    End With
    Set Recordset = Nothing
    Connection.Close
    Set Connection = Nothing
End Sub
```

НА КОМПАКТ-ДИСКЕ

Этот пример, а также база данных Access доступны на прилагаемом к книге компакт-диске.


simple ado example.xls

	A	B	C	D	E	F	G
1	Подразделение	Отдел	Категория	Пункт	Месяц	Бюджет	Выполнено
2	С. Америка	Обработка данных	Оборудование	Рента	Январь	2833	2508
3	С. Америка	Управление персо	Оборудование	Рента	Январь	3565	2500
4	С. Америка	Бухгалтерия	Оборудование	Рента	Январь	3823	3609
5	С. Америка	Обучение	Оборудование	Рента	Январь	3371	2778
6	С. Америка	Безопасность	Оборудование	Рента	Январь	2700	3047
7	С. Америка	Разработка	Оборудование	Рента	Январь	3546	4038
8	С. Америка	Производство	Оборудование	Рента	Январь	2977	3798
9	С. Америка	Продажи	Оборудование	Рента	Январь	4487	2858
10	С. Америка	Снабжение	Оборудование	Рента	Январь	3900	3321
11	С. Америка	Реклама	Оборудование	Рента	Январь	3373	3380
12	С. Америка	Связи с обществе	Оборудование	Рента	Январь	4013	3756
13	С. Америка	Обработка данных	Оборудование	Рента	Февраль	3056	3542
14	С. Америка	Управление персо	Оборудование	Рента	Февраль	2840	2524
15	С. Америка	Бухгалтерия	Оборудование	Рента	Февраль	4010	4490
16	С. Америка	Обучение	Оборудование	Рента	Февраль	3953	3203
17	С. Америка	Безопасность	Оборудование	Рента	Февраль	3693	4066
18	С. Америка	Разработка	Оборудование	Рента	Февраль	3710	3579
19	С. Америка	Производство	Оборудование	Рента	Февраль	2596	2628
20	С. Америка	Продажи	Оборудование	Рента	Февраль	3723	4095
21	С. Америка	Снабжение	Оборудование	Рента	Февраль	3295	3513
22	С. Америка	Реклама	Оборудование	Рента	Февраль	2903	3749
23	С. Америка	Связи с обществе	Оборудование	Рента	Февраль	3906	3595
24	С. Америка	Обработка данных	Оборудование	Рента	Март	2659	4020
25	С. Америка	Управление персо	Оборудование	Рента	Март	3459	3062
26	С. Америка	Бухгалтерия	Оборудование	Рента	Март	4167	3482
27	С. Америка	Обучение	Оборудование	Рента	Март	3064	4165
28	С. Америка	Безопасность	Оборудование	Рента	Март	3031	4487
29	С. Америка	Разработка	Оборудование	Рента	Март	4499	4412
30	С. Америка	Производство	Оборудование	Рента	Март	3219	3937
31	С. Америка	Продажи	Оборудование	Рента	Март	3390	2634

Лист1 Описание

Рис. 20.11. Отображенные данные получены из базы данных Access

Отправка почтовых вложений с помощью Excel

Как вы наверняка знаете, в Excel включены команды отправки рабочих листов и рабочих книг в виде вложений в почтовые сообщения. Конечно, с помощью VBA вы можете автоматизировать подобные операции. Приведенная ниже процедура отправляет активную рабочую книгу (как вложение) по адресу joeblow@anydomain.com. Почтовое сообщение имеет тему Моя книга.

```
Sub SendWorkbook()
     ActiveWorkbook.SendMail "joeblow@anydomain.com", "Моя книга"
End Sub
```

Если требуется отправить только отдельный рабочий лист книги, то вам придется скопировать этот лист в новую (временную) рабочую книгу. После отправки сообщения в виде вложения удалите временный файл. Ниже приведен пример отправки листа Лист1 активной рабочей книги.

```
Sub SendSheet()
     ActiveWorkbook.Worksheets("Лист1").Copy
     ActiveWorkbook.SendMail "joeblow@anydomain.com", "Моя книга"
     ActiveWorkbook.Close False
End Sub
```

В предыдущем примере файлу было присвоено имя рабочей книги по умолчанию (например, Book2.xls). Если вы хотите дать однолистной рабочей книге во вложении более значимое имя, то придется сохранить файл на диске и удалить его после отправки. В следующей процедуре лист Лист1 сохраняется в файле Мой файл.xls. После отправки временной рабочей книги в виде вложения она удаляет с помощью выражения Kill.

```
Sub SendOneSheet()
     Dim Filename As String
     FileName = "Мой файл.xls"
     ActiveWorkbook.Worksheets("Лист1").Copy
     ActiveWorkbookю.SaveAs Filename
     ActiveWorkbook.SendMail "joeblow@anydomain.com", "Моя книга"
     ActiveWorkbook.Close False
Kill Filename
End Sub
```

Использование метода SendKeys

Не все приложения поддерживают автоматизацию. В некоторых случаях вы можете управлять такими приложениями. Для этого воспользуйтесь методом Excel SendKeys, который в результате отправки комбинаций клавиш эмулирует действия пользователя.

Несмотря на то, что использование метода SendKeys является довольно удачным решением, его применение связано с рядом трудностей. Потенциальная проблема заключается в том, что метод применяется по отношению к конкретному пользовательскому интерфейсу. Если более поздняя версия программы имеет другой интерфейс, то приложение, использующее метод SendKeys, перестанет работать. Следовательно, применять метод SendKeys можно только в качестве последнего средства.

Следующий пример демонстрирует применение метода SendKeys по отношению к программе Windows Calculator (Калькулятор). Процедура TestKeys переводит программу в инженерный режим (Команда Вид⇨Инженерный).

```
Sub TestKeys()
    Shell "calc.exe", vbNormalFocus
    AppActivate "Calculator"
    Application.SendKeys "%vs", True
End Sub
```

В данном примере отправляется комбинация клавиш <Alt+V+S> (Символ процента равнозначен клавише <Alt>). Документация по использованию метода `SendKeys` представлена в справочном руководстве, в котором описаны вопросы отправки нестандартных комбинаций клавиш (например, комбинаций с клавишей <Alt>).

Процедура в листинге 20.7 — это более сложный пример использования метода `SendKeys`. В нем также описывается процесс отправки сообщения, но на этот раз с помощью почтового клиента Outlook Express.

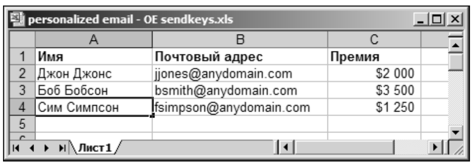
Листинг 20.7. Использование Excel для набора телефонного номера

```
Sub SendEmail()
  Dim cell As Range
  Dim Subj As String
  Dim EmailAddr As String
  Dim Recipient As String
  Dim Bonus As String
  Dim Msg As String
  Dim HLink As String
  For Each cell In Columns("B").Cells.SpecialCells(xlCellTypeConstants)
    If cell.Value Like "*@*" Then
      'Получение данных
      Subj = "Ваша премия"
      Recipient = cell.Offset(0, -1).Value
      EmailAddr = cell.Value
      Bonus = Format(cell.Offset(0, 1).Value, "$0,000.")

     'Составление сообщения
      Msg = "Дорогой " & Recipient & "%0A"
      Msg = Msg & "%0A" & "Рад известить вас о начислении премии в размере "
      Msg = Msg & Bonus & "%0A"
      Msg = Msg & "%0A" & "Вилли Билли"
      Msg = Msg & "%0A" & "Президент"

      'Создание гиперссылки
      HLink = "mailto:" & EmailAddr & "?"
      HLink = HLink & "subject=" & Subj & "&"
      HLink = HLink & "body=" & Msg
      'Send it
      ActiveWorkbook.FollowHyperlink HLink
      Application.Wait (Now + TimeValue("0:00:02"))
      SendKeys "%s", True
    End If
  Next
End Sub
```

На рис. 20.12 показана рабочая книга, которая содержит данные, используемые при создании почтового сообщения.


personalized email - OE sendkeys.xls

	A	B	C
1	Имя	Почтовый адрес	Премия
2	Джон Джонс	jjones@anydomain.com	$2 000
3	Боб Бобсон	bsmith@anydomain.com	$3 500
4	Сим Симпсон	fsimpson@anydomain.com	$1 250
5			

Лист1

Рис. 20.12. Информация, используемая при отправке почтового сообщения

Процедура `SendMail` предполагает, что программа Outlook Express по умолчанию используется в качестве почтового клиента. Она просматривает записи рабочей книги и на основе полученных данных создает сообщение (сохраняется в переменной `Msg`). Для запуска Outlook Express используется метод `FollowHyperlink`. Для первой записи гиперссылка имеет следующий вид.

```
mailto:jjones@anydomain.com?subject=Yuor Annual Bonus&body=Дорогой Джон
Джонсон%0A%0AРад известить вас о начислении премии в размере $2 000.%0A%0AВилли
Билли%0AПрезидент
```



Символы %0A представляют разрыв строки.

Выполнение процедуры приостанавливается на две секунды, а затем используется метод `SendKeys`, который назначает комбинацию клавиш <Alt+S> (с ее помощью отправляется новое почтовое сообщение). Пауза нужна для того, чтобы окно сообщения успело отобразиться перед нажатием комбинации клавиш.

Несмотря на то, что этот метод прекрасно выполняется, обратите внимание на ограничение на длину гиперссылки (около 730 символов). Вы не сможете использовать ее для отправки длинных сообщений.

На рис. 20.13 показано окно сообщения в Outlook Express.

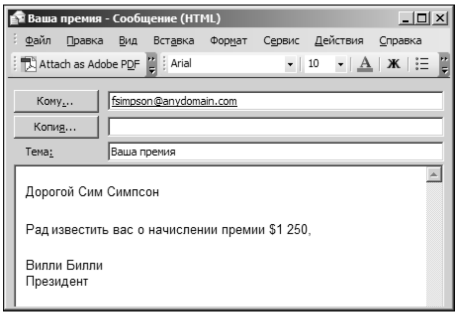


Рис. 20.13. Сообщение Outlook, которое создано на основе данных, сохраненных в рабочей книге

Глава 21

Создание и использование надстроек

В ЭТОЙ ГЛАВЕ...

В этой главе рассмотрены преимущества использования надстроек. Речь пойдет о создании собственных надстроек с помощью инструментов, встроенных в Excel.

- Обзор настроек, заслуживающих внимания разработчиков.
- Дополнительная информация о диспетчере надстроек Excel.
- Пример создания надстройки.
- Различия между файлами XLA и XLS.
- Примеры кода VBA, который используется для управления надстройками.
- Оптимизация надстроек для получения дополнительной производительности и уменьшения размера файла.

Одной из самых полезных возможностей Excel, которой пользуются разработчики, является создание надстроек. Именно надстройки являются подтверждением высокого профессионализма разработчика.

Что такое надстройка

Надстройка — это средство, добавляемое к листу для обеспечения дополнительной функциональности. Например, Excel поставляется с несколькими заранее созданными надстройками. Одной из самых популярных является Пакет анализа. Эта надстройка добавляет в программу инструменты анализа и статистической обработки данных, которые по умолчанию отсутствуют в Excel.

Некоторые надстройки (такие как Пакет анализа) предоставляют новые функции рабочих листов, которые можно добавлять в создаваемые формулы. Новые средства обычно эффективно дополняют исходный интерфейс, а потому становятся неотъемлемой частью программы (этот метод называется *бесшовной интеграцией*).

Сравнение надстройки со стандартной рабочей книгой

Любой опытный пользователь Excel может создать надстройку на основе рабочей книги XLS. Для этого не потребуется дополнительное программное обеспечение и специальные средства. Любой файл XLS можно преобразовать в надстройку, но не любые файлы подходят на роль надстройки. В Excel надстройка представлена обычной рабочей книгой, за небольшим исключением.

- Свойство `IsAddin` объекта `ThisWorkbook` (`ЭтаКнига`) установлено в значение `True`.
- Окно рабочей книги скрыто таким образом, что его нельзя отобразить с помощью команд меню Окно. Это означает, что отобразить рабочую книгу или листы диаграмм надстройки невозможно (кроме случаев создания кода для явного копирования этих объектов в стандартные рабочие книги).
- Надстройка не является членом коллекции `Workbooks` — она входит в коллекцию `AddIns`. При этом сохраняется возможность доступа к надстройке с помощью коллекции `Workbooks` (дополнительная информация приводится в разделе “Членство в коллекциях” далее в этой главе).
- Надстройки можно загружать и выгружать посредством команды Сервис⇨Надстройки.
- Диалоговое окно Макрос не отображает имена макросов, которые содержатся в надстройках.
- Созданные пользователем функции, которые сохранены в надстройке, могут использоваться в формулах без указания имени исходной рабочей книги.

НА ЗАМЕТКУ

По умолчанию надстройки имеют расширение файла `.xla`. Однако это необязательное требование. Файл надстройки может иметь любое разрешение.

Основные причины создания надстройки

Ниже приведены причины, которые могут заставить пользователя конвертировать документ XLS в надстройку.

- *Необходимо ограничить доступ к коду и рабочим листам.* Когда приложение распространяется в виде надстройки, оно защищается с помощью пароля. Чтобы пользователи не смогли получить доступ к листам и модифицировать конфиденциальные методы, следует предотвратить копирование кода или, как минимум, усложнить такую задачу.
- *Необходимо внести ясность.* Если пользователь загружает приложение в виде надстройки, то файл остается невидимым, поэтому вероятность того, что начинающий пользователь на него наткнется, намного снижается. В отличие от скрытых рабочих книг XLS, надстройку невозможно отобразить.
- *Необходимо упростить доступ к функциям рабочего листа.* Создаваемые пользователем функции рабочего листа, заданные в надстройке, не требуют указания имени рабочей книги перед вызовом. Например, если в рабочей книге `Newfuncs` содержится функция `MOVAVG`, то для ее использования в другой рабочей книге необходимо обратиться к следующему синтаксису.

  ```
  =Newfuncs.xls!MOVAVG(A1:A50)
  ```

  Если функция будет храниться в открытой надстройке, то можно использовать более простой синтаксис, так как отсутствует необходимость добавления ссылки на файл.

  ```
  MOVAVG(A1:A50)
  ```

- *Необходимо предоставить пользователям более простой доступ.* Как только расположение надстройки будет указано, она отобразится в диалоговом окне Надстройки с дружественным именем и описанием возможностей.

- *Необходимо получить больший контроль над процессом загрузки.* Надстройки могут открываться автоматически при запуске Excel, независимо от папки, в которой они расположены.
- *Необходимо избежать отображения сообщений при выгрузке.* При закрытии надстройки не отображаются сообщения "Сохранить изменения в xxx?".

О надстройках COM

Excel 2000 и более поздние версии программы поддерживают надстройки COM (Component Object Model). Эти файлы имеют расширение `DLL` или `EXE`. Надстройки COM создаются таким образом, что работают во всех приложениях Office, поддерживающих эту возможность. Дополнительным преимуществом является компилируемость кода, что увеличивает безопасность. В отличие от надстроек XLA, надстройки COM не могут содержать листы или диаграммы Excel. Надстройки COM разрабатываются с помощью Visual Basic 5 (и более поздних версий) или пакета Office Developer Edition. Изучение методов создания надстроек COM выходит за пределы вопросов, рассматриваемых в этой книге.

Использование диспетчера надстроек Excel

Наиболее эффективным способом загрузки и выгрузки надстроек является использование диалогового окна Excel Надстройки, доступ к которому можно получить с помощью команды Сервис⇨Надстройки. Эта команда приведет к отображению диалогового окна Надстройки, которое показано на рис. 21.1. Область списка содержит имена всех надстроек, которые известны Excel. Кроме того, флажки указывают на уже открытые надстройки. Надстройки можно подключать и отключать только в этом диалоговом окне. Вам необходимо установить или сбросить флажок напротив представленной в списке надстройки.

СОВЕТ

Диалоговое окно Надстройки в Excel 2002–2003 имеет кнопку Автоматизация. Эта кнопка используется для установки надстроек COM. Несмотря на то, что Excel 2000 поддерживает надстройки COM, непосредственного способа их установки эта версия не предоставляет.

ВНИМАНИЕ

Большинство файлов надстроек также можно открыть с помощью команды Файл⇨Открыть. Так как надстройка никогда не станет активной рабочей книгой, ее невозможно закрыть с помощью команды Файл⇨Закрыть. Надстройка удаляется только в результате перезапуска Excel или выполнения кода VBA, который закрывает надстройку. Подключение надстройки с помощью команды Файл⇨Открыть приводит к открытию файла, но официально надстройка не считается "установленной".

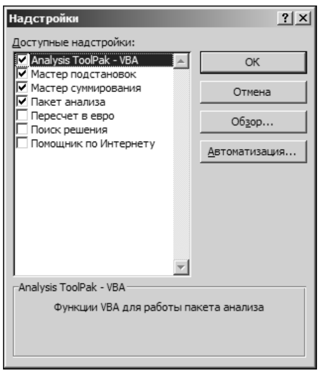


Рис. 21.1. Диалоговое окно Надстройки

При открытии надстройки новые возможности, а также инструменты можно и не заметить. В большинстве случаев пользовательский интерфейс незначительно, но изменяется: Excel отображает новый элемент меню или новую панель инструментов. Например, если открыть надстройку Пакет анализа, то появится новая команда Сервис⇨Анализ данных. При открытии надстройки Пересчет в евро появится новая панель инструментов EuroValue. Если надстройка содержит только пользовательские функции рабочего листа, то новые функции будут добавлены в диалоговое окно Мастер функций.

Создание надстройки

Как отмечалось ранее, в надстройку можно превратить любую рабочую книгу, но не все рабочие книги являются подходящим материалом для создания надстройки. От превращения в надстройку более всего "выигрывают" рабочие книги, содержащие макросы, особенно те, которые включают процедуры универсального назначения. Рабочая книга, которая состоит только из рабочих листов, окажется недоступной после создания надстройки, так как листы в надстройке скрываются от пользователя. Однако вы можете создать код, который будет копировать все или часть листов из надстройки в отображаемую на экране рабочую книгу.

Создание надстройки на основе рабочей книги — это очень простая задача. Следующие инструкции описывают создание надстройки на основе файла стандартной рабочей книги.

1. Создайте приложение и удостоверьтесь, что оно работает правильно.

   Не забудьте определить способ запуска макросов (или макроса), которые содержатся в надстройке. Может возникнуть необходимость в добавлении элемента меню, нескольких опций меню или дополнительной панели инструментов. В главе 23 изложена дополнительная информация о модификации меню, а в главе 22 содержатся сведения о создании собственных панелей управления.

2. Протестируйте приложение, запустив его в тот момент, когда активной является *другая* рабочая книга.

   Таким образом, будет проверено поведение приложения в качестве надстройки (так как надстройка никогда не является активной рабочей книгой).

3. Запустите редактор VBE и выберите рабочую книгу в окне Project. Выполните команду Tools⇨xxx Properties (Сервис⇨Свойства xxx), после чего щелкните на вкладке Protection (Защита). Установите флажок Lock project for viewing (Блокировать проект для просмотра) и введите пароль (дважды). После этого щелкните на кнопке OK.

   Этот шаг необходим для того, чтобы ограничить доступ конечных пользователей к исходному коду макросов и диалоговых окон.

Несколько слов о безопасности

Microsoft никогда не рекламировала Excel как программный продукт, который предназначен для создания приложений с защищенным исходным кодом. Возможность добавления пароля в Excel представлена для ограничения доступа обычных пользователей к определенным фрагментам приложения. Excel 2003 предоставила более высокий уровень безопасности, по сравнению с предыдущими версиями, но вероятность взлома установленной защиты все еще велика. Если вы не хотите, чтобы кто-либо видел исходный код или формулы приложения, то не ст*о*ит избирать Excel в качестве инструмента разработки.

---

4. Повторно активизируйте Excel и выберите команду Файл⇨Свойства. Перейдите на вкладку Документ и введите краткий описательный заголовок в поле Название. Более полное описание можно ввести в поле Заметки.

   Этот шаг не является обязательным, но делает надстройку проще в использовании, так как введенный текст описания отображается в диалоговом окне Надстройки.

5. Выполните команду Файл⇨Сохранить как.
6. В диалоговом окне Сохранение документа выберите Надстройка Microsoft Excel в раскрывающемся списке Тип файла.
7. Щелкните на кнопке Сохранить. Копия рабочей книги будет сохранена в виде файла с расширением `*.xla`. Первоначальная рабочая книга будет оставаться открытой.

Рабочая книга, которая превращается в надстройку, должна содержать как минимум один рабочий лист. Например, если в рабочей книге расположены только листы диаграмм или диалоговые листы Excel 5/95, то в диалоговом окне Сохранение документа нельзя выбрать тип файла Надстройка Microsoft Excel. Более того, данный тип файла доступен только тогда, когда при выборе команды Файл⇨Сохранить как активен рабочий лист.

Пример надстройки

В этом разделе будут рассмотрены инструкции по созданию надстройки (надстройка получена на основе утилиты Text Tools, которая рассматривалась в главе 16).

XLS-версия утилиты Text Tools доступна на прилагаемом к книге компакт диске. Можно использовать этот файл для создания описанной ниже надстройки.

Настройка рабочей книги

В приведенном далее примере будет использоваться уже отлаженная рабочая книга, состоящая из нескольких элементов.

- Рабочий лист, который называется `Лист1`. Этот лист содержит задаваемые пользователем данные, которые восстанавливаются в случае отмены выполненных операций.
- Пользовательское диалоговое окно `UserForm1`. Применяется для предоставления основного пользовательского интерфейса. Модуль кода для данного диалогового окна содержит несколько процедур обработки событий.
- Модуль кода VBA, который называется `Module1`. Этот модуль содержит несколько процедур, включая процедуру отображения диалогового окна `FormMain`.
- Кроме того, в модуле кода `ThisWorkbook` находятся две процедуры обработки событий (`Workbook_Open` и `Workbook_BeforeClose`), которые вызывают другие процедуры для создания и удаления опций меню.

В главе 16 изложена дополнительная информация о принципах работы утилиты Text Tools.

Тестирование рабочей книги

Перед преобразованием этой рабочей книги в надстройку первую необходимо протестировать. Для создания условий, в которых будет работать книга после преобразования в надстройку, проверку ее работоспособности следует провести тогда, когда активной является другая рабочая книга.

Откройте новую рабочую книгу и воспользуйтесь некоторыми возможностями утилиты Text Tools. Выполните любые действия, чтобы заставить утилиту выдать сообщение об ошибке. Более удачный вариант состоит в следующем — пригласите пользователя, который никогда ранее не работал с этим приложением. Необходимо, чтобы он содействовал неудачному завершению работы приложения.

Добавление описательной информации

Рекомендуется всегда добавлять к надстройке описание. Выполните команду Файл⇨Свойства, чтобы открыть диалоговое окно Свойства. После этого перейдите на вкладку Документ, которая показана на рис. 21.2.

Введите заголовок надстройки в поле Название. Этот текст представлен в диалоговом окне Надстройки. В поле Заметки необходимо ввести более полное описание надстройки. Эта информация будет отображена в нижней части диалогового окна Надстройки при выборе этой надстройки.

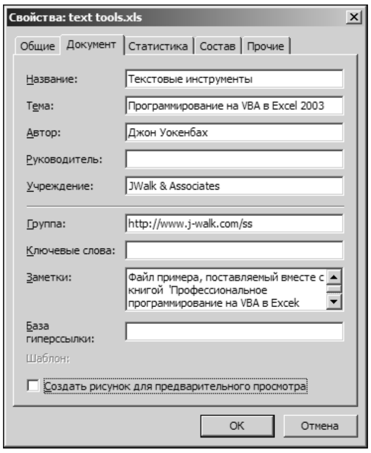


Рис. 21.2. Воспользуйтесь диалоговым окном Свойства для введения информации, описывающей надстройку

Создание надстройки

Для того чтобы создать надстройку, выполните следующие действия.

1. Активизируйте редактор VBE и выберите рабочую книгу в окне проектов.
2. Выполните команду Debug⇨Compile (Отладка⇨Компилировать). Этот шаг приведет к компиляции кода VBA, что позволит обнаружить явные ошибки в исходном коде. После сохранения файла в виде надстройки Excel создаст надстройку даже в том случае, если исходный код содержит ошибки.
3. Выберите команду Tools⇨TextTools Properties (Сервис⇨Свойства Text Tools) и перейдите на вкладку Protection (Защита) в появившемся диалоговом окне. Установите флажок Lock project for viewing (Блокировать просмотр проекта) и введите пароль (дважды). Щелкните на кнопке OK. Если необходимость в защите проекта отсутствует, то этот шаг можно опустить.

4. По умолчанию все проекты VBA называются `VBProject`. В этом примере имя проекта изменено на `TextTools`.
5. Сохраните рабочую книгу.
6. Активизируйте рабочую книгу и выполните команду Файл⇨Сохранить как. Excel отобразит диалоговое окно Сохранение документа.
7. В раскрывающемся списке Тип файла выберите Надстройка Microsoft Excel.
8. Щелкните на кнопке Сохранить. Будет создана новая надстройка, при этом первоначальная версия рабочей книги останется открытой.

О диспетчере надстроек Excel

Доступ к диспетчеру надстроек Excel осуществляется с помощью команды Сервис⇨Надстройки, которая отображает диалоговое окно Надстройки. Это диалоговое окно содержит имена всех доступных надстроек. Те надстройки, напротив имен которых установлены флажки, являются открытыми.

В терминах VBA диалоговое окно Надстройки перечисляет значения свойства `Title` каждого объекта `AddIn` коллекции `AddIns`. Каждая надстройка, для которой установлен флажок, имеет свойство `Installed` со значением `True`.

Надстройку можно установить, щелкнув на флажке возле ее имени. Сбросив этот флажок, вы выгрузите выбранную надстройку. Для того чтобы добавить в список новую надстройку, необходимо воспользоваться кнопкой Обзор и указать расположение файла надстройки. По умолчанию диалоговое окно отображает файлы следующих типов.

- XLA: надстройка, созданная на основе файла XLS.
- XLL: отдельный файл DLL, который создан с помощью компилируемого языка C.

Если щелкнуть на кнопке Автоматизация (которая отображается только в Excel 2002–2003), то вам будет предоставлена возможность просмотреть список надстроек COM. Обратите внимание, что диалоговое окно Серверы автоматизации содержит большое количество файлов, но не все они являются надстройками COM, которые предназначены для работы в Excel.

Файл надстройки можно добавить в коллекцию `AddIns` с помощью метода `Add` этой коллекции, но удалить такой файл из коллекции с помощью VBA нельзя. Кроме того, надстройку можно открыть, если присвоить свойству `Installed` объекта `AddIn` значение `True`. Установка этого свойства в значение `False` приведет к выгрузке надстройки.

При выходе из диспетчера надстроек Excel он сохраняет информацию об установленных надстройках в системном реестре. Таким образом, надстройки, которые установлены в момент выхода из Excel, автоматически открываются при следующем запуске Excel.

Установка надстройки

Во избежание путаницы, закройте рабочую книгу XLS перед установкой надстройки, создаваемой на основе этой рабочей книги.

Чтобы установить надстройку, выполните следующие действия.

1. Выполните команду Сервис⇨Надстройки. Excel отобразит диалоговое окно Надстройки.
2. Щелкните на кнопке Обзор и найдите файл надстройки, которая была только что создана. После указания расположения файла надстройки диалоговое окно Надстройки добавит надстройку в список. Как показано на рис. 21.3, диалоговое

окно Надстройки отображает также и описание, которое введено при создании надстройки в диалоговом окне Свойства.

3. Щелкните на кнопке OK, чтобы закрыть диалоговое окно и открыть выбранную надстройку.

После открытия надстройки Text Tools меню Сервис будет содержать новый элемент, который запускает процедуру `StartTextTools`, хранящуюся в надстройке.

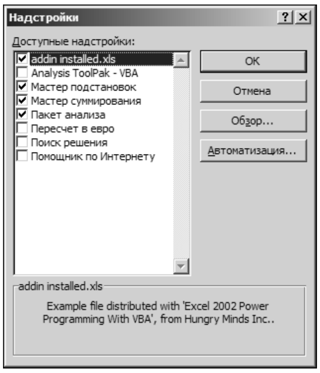


Рис. 21.3. Диалоговое окно Надстройки, в котором выбрана новая надстройка

Распространение надстройки

Эту надстройку можно распространять среди других пользователей Excel, передавая им копию файла XLA (версия XLS им не понадобится). Также пользователям необходимо передать инструкции по установке надстройки. После установки надстройки в меню Сервис будет содержаться новая команда Текстовые инструменты. Если файл защищен паролем, то код макроса не смогут просмотреть и модифицировать посторонние пользователи. Исключение составляют те пользователи, которым известен пароль.

Изменение надстройки

Если в надстройку необходимо внести изменения, то ее вначале следует открыть, а после этого разблокировать. Для того чтобы разблокировать надстройку, активизируйте редактор VBE и дважды щелкните на имени проекта в окне Project. После этого на экране появится запрос на введение пароля. Внесите необходимые изменения и сохраните файл, не покидая редактор VBE (для этого выберите команду File⇨Save (Файл⇨Сохранить)).

Если создается надстройка, которая хранит данные на рабочем листе, потребуется установить свойство `IsAddIn` рабочей книги в значение `False`. Это необходимо сделать перед просмотром рабочей книги в Excel. Свойство изменяется в диалоговом окне Properties при выделенном объекте `ThisWorkbook` (рис. 21.4). После внесения изменений удостоверьтесь, что свойство `IsAddIn` опять установлено в положение `True`. Если перед сохранением файла оставить это свойство установленным в значение `False`, то файл будет сохранен как обычная рабочая книга (хотя он имеет расширение `.xla`). На данном этапе попытка использовать файл в качестве надстройки с помощью диалогового окна Надстройки приведет к отображению сообщения об ошибке.

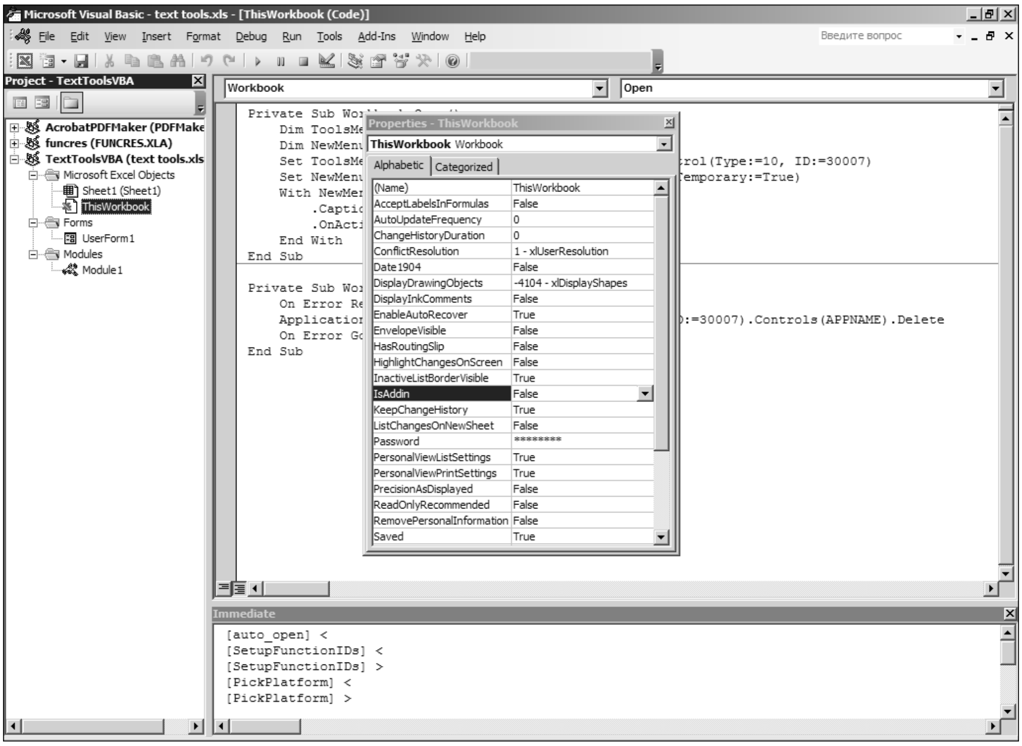


Рис. 21.4. Создание надстройки

Список действий, необходимых для создания надстройки

Перед распространением надстройки осуществите проверку соответствия произведенных действий следующему списку.

- Проверена ли надстройка на всех поддерживаемых платформах и версиях Excel?
- Присвоено ли проекту VBA новое имя? По умолчанию каждый проект называется `VBAProject`. Рекомендуется присваивать проекту описательное имя.
- Анализирует ли надстройка структуру и названия папок компьютера?
- Корректно ли отображается имя надстройки и ее описание при использовании диалогового окна Надстройки?
- Если надстройка содержит функции VBA, которые не предназначены для использования в рабочей книге, были ли функции объявлены с областью действия `Private`? Если это не так, то такие функции будут отображены в диалоговом окне Мастер функций.
- Выполнялась ли принудительная повторная компиляция надстройки, позволяющая удостовериться в отсутствии синтаксических ошибок?
- Учитывались ли вопросы межнационального использования? Например, если надстройка создает новый элемент меню Сервис, то процедура может завершиться сообщением об ошибке (меню Сервис должно иметь английское имя).
- Оптимизирована ли надстройка на максимальное быстродействие? Дополнительная информация по этой теме приводится в разделе “Оптимизация производительности надстроек”.

Сравнение файлов XLA и XLS

Итак, сравним файл надстройки XLA с исходным файлом XLS и рассмотрим методы оптимизации производительности надстройки. Речь пойдет о способах уменьшения размеров файлов, что позволит сократить время загрузки и уменьшить объем используемого дискового пространства и оперативной памяти.

Структура и размер файлов

Надстройка, основанная на файле XLS, имеет такой же размер, как и исходный файл рабочей книги. Код VBA файла XLA не сжимается и не оптимизируется, поэтому увеличение быстродействия не относится к достоинствам надстроек.

Членство в коллекциях

Надстройка является членом коллекции `AddIns`, но не выступает “официальным” членом коллекции `Workbooks`. К надстройке можно обращаться как к элементу коллекции `Workbooks`, указав в качестве индекса имя файла, в котором хранится надстройка. Следующий оператор создает объект, который представляет надстройку `Myaddin.xla`.

```
Set TestAddin = Workbook("Myaddin.xla")
```

На надстройки не допускается ссылаться с помощью числового индекса коллекции `Workbooks`. Если применить приведенный далее код для просмотра членов коллекции `Workbooks`, то надстройка `Myaddin.xla` отображена не будет.

```
For Each w in Application.Workbooks
    MsgBox w.Name
Next w
```

С другой стороны, следующий цикл `For Next` позволяет отобразить надстройку `Myaddin.xla` (если, конечно, Excel “знает” о ней) в диалоговом окне Надстройки.

```
For Each a in Application.AddIns
    MsgBox a.Name
Next a
```

Окна

Обычные рабочие книги XLS отображаются в одном или нескольких окнах. Например, представленный далее оператор отображает количество окон активной рабочей книги.

```
MsgBox ActiveWorkbook.Windows.Count
```

Состоянием отображения каждого окна рабочей книги XLS можно управлять с помощью команды Окно⇨Скрыть или посредством значений свойства `Visible`. Следующий код скрывает все окна активной рабочей книги.

```
For Each Win In ActiveWorkbook.Windows
    Win.Visible = False
Next Win
```

Файлы надстроек никогда не отображаются и официально не имеют окон, хотя содержат рабочие листы (скрытые). Следовательно, в списке открытых окон меню Окно надстройки не представлены. Если надстройка `Myaddin.xla` открыта, то следующий оператор возвратит значение 0.

```
MsgBox Workbooks("Myaddin.xla").Windows.Count
```

Листы

Файлы надстроек XLS, как и файлы XLS, могут содержать любое количество рабочих листов или листов диаграмм. Но, как было отмечено ранее в этой главе, файл XLS должен иметь как минимум один рабочий лист, чтобы можно было преобразовать его в надстройку.

После открытия надстройки код VBA будет получать доступ к листам, содержащимся в надстройке, как и в случае обычной рабочей книги. По причине того, что файлы надстроек не являются частью коллекции `Workbooks`, на надстройку необходимо ссылаться по имени, а не по индексному номеру. Из следующего примера вы узнаете, как извлекать значения ячейки `A1` на первом рабочем листе надстройки `Myaddin.xla`, открытой в момент выполнения этого кода.

```
MsgBox Workbooks("Myaddin.xla").Worksheets(1) _
    .Range("A1").Value
```

Если надстройка содержит рабочий лист, который должен отображаться, то можно скопировать его в открытую рабочую книгу или создать новую рабочую книгу на основе этого листа.

Например, представленный ниже код копирует первый рабочий лист надстройки и размещает его в активной рабочей книге (в качестве последнего листа рабочей книги).

```
Sub CopySheetFromAddin()
    Set AddinSheet = Workbooks("Myaddin.xla").Sheets(1)
    NumSheets = ActiveWorkbook.Sheets.Count
    AddinSheet.Copy After:=ActiveWorkbook.Sheets(NumSheets)
End Sub
```

Создание рабочей книги на основе листа из надстройки осуществляется еще проще.

```
Sub CopySheetFromAddin()
    Workbooks("Myaddin.xla").Sheets(1).Copy
End Sub
```

НА ЗАМЕТКУ

Предыдущий пример предполагает, что код находится в файле, отличном от файла надстройки. Код VBA в файле надстройки всегда может использовать объект `ThisWorkbook`, чтобы задать ссылки на листы или диапазоны надстройки. Например, следующий оператор должен находиться в модуле кода VBA надстройки. Следующий оператор отображает значение ячейки `A1` первого листа надстройки.

```
MsgBox ThisWorkbook.Sheets("Лист1").Range("A1").Value
```

Получение доступа к VBA-процедурам надстройки

Доступ к процедурам VBA, которые сохраняются в надстройке, несколько отличается от доступа к процедурам файла обычной рабочей книги XLS. Если выполнить команду Сервис⇨Макрос⇨Макросы, то диалоговое окно Макрос не будет содержать макросы, которые сохранены в открытой надстройке. Это вызвано попыткой Excel ограничить доступ к таким процедурам.

СОВЕТ

Если имя процедуры известно, то его можно ввести непосредственно в диалоговом окне Макрос, после чего следует щелкнуть на кнопке Выполнить — процедура будет запущена. При этом процедура `Sub` должна находиться в модуле кода общего назначения, а не в модулях кода конкретных объектов.

По причине того, что процедуры надстройки не отображаются в диалоговом окне Макрос, необходимо обеспечить другие способы обращения к ним. В число возможных вариантов решения проблемы входит использование стандартных методов: назначение комбинаций клавиш, опций меню и кнопок пользовательских панелей инструментов; а также косвенных методов (создание процедур обработки событий). Одним из главных выступает метод `OnTime`, который выполняет процедуру в определенное время суток.

Можно также воспользоваться методом `Run` объекта `Application` для выполнения процедуры, содержащейся в надстройке.

```
Application.Run "Myaddin.xla!DisplayNames"
```

Еще одним вариантом можно считать использование команды Tools⇨References редактора VBE, которая позволяет создать ссылку на надстройку. В итоге вы непосредственно ссылаетесь на одну из процедур в коде VBA, не указывая имени файла. На самом деле необходимости в использовании метода `Run` не существует: процедуру можно вызывать непосредственно, пока она не объявлена с ключевым словом `Private`. Представленный далее оператор выполняет процедуру `DisplayNames`, которая содержится в надстройке, указанной с помощью команды Tools⇨References.

```
Call DisplayNames
```

Даже после создания ссылки на надстройку имена процедур не отображаются в диалоговом окне Макрос.

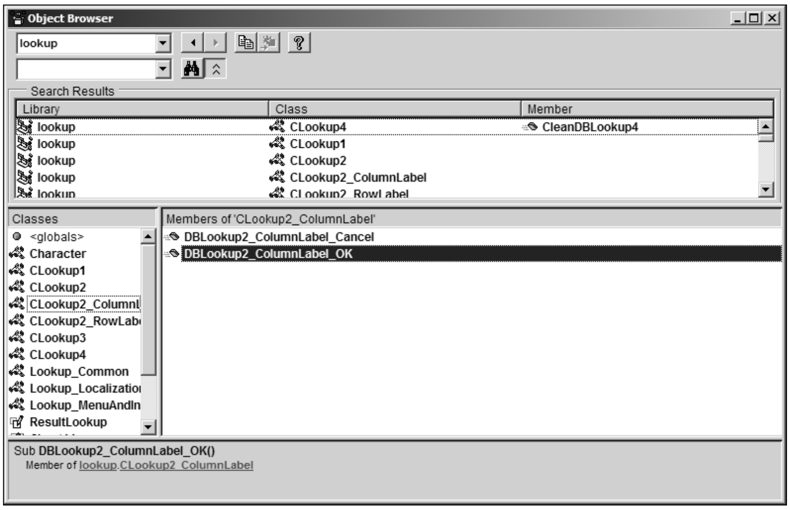
Функции определяются в надстройке подобно тому, как это осуществляется в рабочей книге XLS. К ним легко обращаться, поскольку Excel отображает имена функций в диалоговом окне Мастер функций (в категории Определенные пользователем). Единственным исключением являются функции, объявленные как `Private`. После такого объявления функция не будет отображаться в диалоговом окне Мастер функций. Именно поэтому объявление пользовательских функций с областью действия `Private` всегда оправдано, когда их необходимо использовать только в процедурах, но не в формулах рабочих листов.

Поиск в защищенной надстройке

Диалоговое окно Макрос не отображает имена процедур, которые содержатся в надстройках. Однако порой требуется запустить подобную процедуру, когда надстройка защищена (что не позволит обратиться к исходному коду для определения имени процедуры). Для решения подобной проблемы воспользуйтесь окном Object Browser.

Чтобы продемонстрировать этот способ, выполните команду Сервис⇨Надстройки для установки надстройки Мастер подстановок. Эта надстройка распространяется вместе с Excel. Она защищена от просмотра, что не позволяет просматривать исходный код ее макросов.

1. Активизируйте редактор VBE и выберите проект `Lookup.xla` в окне Project.
2. Нажмите клавишу <F2> для активизации окна Object Browser.
3. В раскрывающемся списке Libraries (Библиотеки) выберите lookup. Это приведет к отображению всех классов надстройки Мастер подстановок.
4. Выберите различные элементы в списке Classes (Классы), чтобы ознакомиться с классами и их элементами.



Например, класс `Lookup_Common` является модулем и содержит целый ряд переменных, констант, процедур и функций. Одна из этих процедур `DoLookupCommand` выглядит как процедура, которая используется для запуска мастера. Для того чтобы проверить изложенную теорию, активизируйте Excel и выберите команду Сервис⇨Макрос⇨Макросы. Введите `DoLookupCommand` в поле Имя макроса и щелкните на кнопке Выполнить. Как и ожидалось, будет отображено первое диалоговое окно мастера подстановок.

После получения этой информации можно приступить к созданию кода VBA, который будет запускать мастер подстановок.

Если необходимо просмотреть надстройку, которая не объявляет свои функции как `Private`, установите надстройку Мастер подстановок. После этого щелкните на кнопке Вставка функции. В разделе Определенные пользователем будет отображен список, который состоит из более чем десяти функций, не предназначенных для использования на рабочих листах. Эти функции не могут быть вставлены в формулы, находящиеся на рабочих листах.

Как отмечалось ранее, можно использовать функции рабочих листов, которые находятся в надстройке, не указывая имени рабочей книги. Например, если в файле `Newfuncs.xls` хранится функция `MOVAVG`, то необходимо использовать следующий оператор для обращения к функции из другой рабочей книги.

```
=Newfuncs.xls!MOVAVG(A1:A50)
```

Однако функция зачастую хранится в открытом файле надстройки. Тогда опустите ссылку на файл и используйте более простой синтаксис.

```
=MOVAVG(A1:A50)
```

Управление надстройками с помощью кода VBA

В этом разделе предоставляется информация, которая поможет создавать процедуры VBA, используемые для управления надстройками.

Коллекция AddIns

Коллекция `AddIns` состоит из объектов всех надстроек, о которых “знает” Excel. В нее входят как установленные, так и неустановленные надстройки. Команда Сервис⇨Надстройки отображает диалоговое окно Надстройки, в котором перечислены все члены коллекции `AddIns`. Возле названий надстроек расположены флажки, выставленные для установленных надстроек.

ДОБАВЛЕНИЕ ЭЛЕМЕНТА В КОЛЛЕКЦИЮ ADDINS

Файлы надстроек, составляющие коллекцию `AddIns`, могут находиться в любой папке. Excel сохраняет неполный список файлов и папок, в которых они находятся, в системном реестре Windows. Excel 2003 хранит эту информацию по следующему пути.

```
HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Excel\Add-in Manager
```

Можно воспользоваться редактором реестра Windows (`regedit.exe`) для просмотра соответствующих параметров реестра. Обратите внимание на то, что “стандартные” надстройки, которые поставляются вместе с Excel, не указаны в этой записи реестра. Кроме того, файлы надстроек, которые хранятся в указанной ниже папке, тоже будут добавлены в список (хотя и не будут указаны в системном реестре).

```
Windows\Application data\Microsoft\Addins
```

Вы можете добавить новый объект `AddIn` в коллекцию `AddIns` как вручную, так и программно с помощью VBA. Для того чтобы вручную добавить новую надстройку в коллекцию, выберите Сервис⇨Надстройки, щелкните на кнопке Обзор и найдите файл необходимой надстройки.

Добавить новый член в коллекцию `AddIns` с помощью VBA можно, воспользовавшись методом `Add` коллекции `AddIns`.

```
Application.AddIns.Add "c:\files\newaddin.xla"
```

После выполнения представленной выше операции коллекция `AddIns` будет содержать дополнительный элемент, а диалоговое окно Надстройки отобразит новый элемент списка. Если надстройка уже существует в коллекции, то ничего не произойдет, и сообщение об ошибке не будет генерироваться.

Когда добавляемая надстройка находится на сменном носителе (например, на дискете или на компакт-диске), вы имеете возможность скопировать файл в папку библиотеки Excel с помощью метода `Add`. В следующем примере файл `Myaddin.xla` копируется с диска `A:` и добавляется в коллекцию `AddIns`. Второй аргумент (в данном случае равный `True`) указывает на необходимость копирования надстройки. Если надстройка находится на жестком диске, то второй аргумент можно игнорировать.

```
Application.AddIns.Add "a:\Myaddin.xla", True
```

Добавление новой надстройки в коллекцию `AddIns` не приводит к ее установке. Для того чтобы установить надстройку, необходимо присвоить ее свойству `Installed` значение `True`.

Системный реестр не обновляется до того момента, пока Excel не завершит свою работу привычным способом. Если Excel завершает свою работу анормально (т.е. со сбоем), то имя надстройки не будет добавлено в системный реестр, а надстройка не станет частью коллекции `AddIns` при следующем запуске Excel.

УДАЛЕНИЕ ЭЛЕМЕНТА ИЗ КОЛЛЕКЦИИ ADDINS

Как ни удивительно, однако не существует явного способа удаления элемента из коллекции `AddIns`. Коллекция `AddIns` не имеет метода `Delete` или `Remove`. Одним из способов удаления надстройки из диалогового окна Надстройки является непосредственное редактирование параметра системного реестра (для этого используется программа `regedit.exe`). В результате надстройка не будет отображена в списке диалогового окна Надстройки при следующем запуске Excel. Обратите внимание на то, что данный метод не применим ко всем файлам надстроек.

Еще одним способом удаления надстройки из коллекции `AddIns` является удаление, перемещение или переименование файла XLA, который содержит надстройку. При последующей попытке установки этой надстройки будет отображено предупреждение, которое показано на рис. 21.5. В этом предупреждении пользователю предоставляется возможность удалить надстройку из коллекции `AddIns`.

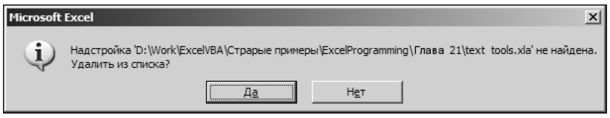


Рис. 21.5. Такой хитроумный способ удаления надстройки из коллекции `AddIns` весьма эффективен

Свойства объекта AddIn

Объект `AddIn` является единственным представителем коллекции `AddIns`. Например, для того чтобы отобразить имя файла, в котором содержится надстройка, являющаяся первым членом коллекции `AddIns`, можно воспользоваться следующим оператором.

```
MsgBox AddIns(1).Name
```

Объект `AddIn` имеет 14 свойств, информацию о которых можно получить в справочном руководстве. Пять свойств являются “скрытыми”. Названия таких свойств довольно запутаны, поэтому мы рассмотрим некоторые наиболее важные свойства этого объекта.

NAME

Это свойство содержит имя файла надстройки. `Name` — свойство, предназначенное только для чтения, оно не позволяет изменить имя файла и указать другое значение свойства `Name`.

PATH

Свойство, содержащее имя диска и папки, в которых расположен файл надстройки. Значение этого свойства не имеет в конце наклонной черты и имени файла.

FULLNAME

Это свойство содержит полное имя надстройки, которое состоит из имени диска, пути и имени файла. Данное свойство бесполезное, поскольку такая информация

предоставляется свойствами Name и Path. Следующие операторы в результате выполнения возвращают одинаковые строки.

```
MsgBox AddIns(1).Path & "\" & AddIns(1).Name
MsgBox AddIns(1).FullName
```

TITLE

Скрытое свойство, содержащее описательное имя надстройки. Свойство Title отображается в диалоговом окне Надстройки. Данное свойство предназначено только для чтения, и единственный способ изменить значение этого свойства — использовать команду Файл⇨Свойства (перейдите на вкладку Документ и введите текст в поле Название). Эту команду необходимо выполнять по отношению к файлу XLS до того, как он будет преобразован в надстройку.

Как правило, к члену коллекции обращаются с помощью значения свойства Name. Коллекция AddIns этим отличается от других коллекций. В ней вместо свойства Name для адресации используется значение свойства Title. Следующий пример отображает имя файла надстройки View Manager (т.е. View.xls), значение свойства Title которой равно View Manager.

```
Sub ShowName()
    MsgBox AddIns("View Manager").Name
End Sub
```

Можно, конечно, ссылаться на определенную надстройку по индексу, если индекс известен точно.

COMMENTS

Это скрытое свойство хранит текст, который отображается в диалоговом окне Надстройки, описывающем определенную надстройку. Свойство Comments предназначено только для чтения. Единственный способ изменить его значение — использовать диалоговое окно Свойства до преобразования рабочей книги в надстройку. Описание может достигать 255 символов, но диалоговое окно Надстройки отображает из них только первые 100.

INSTALLED

Свойство Installed имеет значение True, если надстройка установлена (т.е. в диалоговом окне Надстройки напротив имени надстройки установлен флажок). Установка свойства Installed в значение True приводит к загрузке надстройки. Установка этого свойства в значение False приводит к выгрузке надстройки из памяти. Ниже приведен пример установки (т.е. открытия) надстройки MS Query с помощью кода VBA.

```
Sub InstallQuery()
    AddIns("MS Query").Installed = True
End Sub
```

После выполнения процедуры диалоговое окно Надстройки будет отображать установленный флажок напротив имени надстройки MS Query. Если надстройка уже установлена, то присвоение свойству Installed значения True не приведет к выполнению каких-либо действий. Для того чтобы отключить надстройку (удалить), установите свойство Installed в значение False.

> ВНИМАНИЕ: Если надстройка открыта с помощью команды Файл⇨Открыть, то она не считается установленной. Следовательно, ее свойство Installed будет иметь значение False.

Следующая процедура отображает количество надстроек, которые находятся в коллекции `AddIns`, и количество надстроек, которые установлены в данный момент. Легко заметить, что при подсчете установленных надстроек не учитываются надстройки, которые открыты с помощью команды Файл⇨Открыть.

```
Sub CountInstalledAddIns()
    Dim Count As Integer
    Dim Item As AddIn
    Dim Msg As String
    Count = 0
    For Each Item In AddIns
        If Item.Installed Then Count = Count + 1
    Next Item
    Msg = "Надстройки: " & AddIns.Count & Chr(13)
    Msg = Msg & "Установлено: " & Count
    MsgBox Msg
End Sub
```

Следующая процедура анализирует все надстройки в коллекции `AddIns` и удаляет из памяти все надстройки, установленные в данный момент. Данная процедура не влияет на надстройки, которые открыты с помощью команды Файл⇨Открыть.

```
Sub InstallAll()
    Dim Count As Integer
    Dim Item As AddIn
    Count = 0
    For Each Item In AddIns
        If Item.Installed Then
           Item.Installed = False
           Count = Count + 1
        End If
    Next Item
    MsgBox "Выгружено " & Count & " надстроек"
End Sub
```

С помощью значения свойства `IsAddIn` можно определить, является ли рабочая книга надстройкой. Это свойство предназначено как для чтения, так и для записи, поэтому рабочую книгу можно превратить в надстройку в результате установки свойства `IsAddIn` в значение `True`.

ПОЛУЧЕНИЕ ДОСТУПА К НАДСТРОЙКЕ КАК К РАБОЧЕЙ КНИГЕ

Как отмечалось ранее, существует два метода открытия надстройки: с помощью команды Файл⇨Открыть и с помощью команды Сервис⇨Надстройки. Второй метод более предпочтителен при управлении надстройками. На то существует особая причина: при открытии надстройки с помощью команды Файл⇨Открыть свойство `Installed` *не* устанавливается в значение `True`. Таким образом, данный файл невозможно закрыть с помощью диалогового окна Надстройки. Единственной возможностью закрыть эту надстройку является использование соответствующего оператора VBA. Например, можно применить оператор

```
Workbooks("Myaddin.xla").Close
```

Использование метода `Close` по отношению к установленной надстройке приводит к удалению надстройки из памяти, но *не* устанавливает ее свойство `Installed` в значение `False`. Таким образом, диалоговое окно Надстройки все еще будет указывать эту надстройку в числе установленных, что может ввести пользователя в заблуждение. Правильным способом удаления установленных надстроек является установка их свойства `Installed` в значение `False`.

Итак, как вы заметили, что возможность использования надстроек Excel является несколько неоднозначной. Разработчик должен быть особо внимателен к вопросам установки и удаления надстроек.

События объекта AddIn

Объект `AddIn` имеет два события: `AddInInstall` (событие возникает при установке надстройки) и `AddInUninstall` (событие возникает при удалении надстройки). Можно создать процедуры обработки этих событий, которые размещаются в модуле кода объекта `ThisWorkbook`, соответствующего надстройке.

Следующий пример отображает окно сообщения в момент установки надстройки.

```
Private Sub Workbook_AddinInstall()
    MsgBox ThisWorkbook.Name & _
        " установлена"
End Sub
```

Не путайте событие `AddInInstall` и событие `Open`. Событие `AddInInstall` возникает только при первой установке надстройки. Если необходимо выполнять определенный код каждый раз при открытии надстройки, то можно воспользоваться процедурой `Workbook_Open`.

Дополнительная информация о событиях приведена в главе 19.

Оптимизация производительности надстроек

Основное требование к надстройкам — это максимальная производительность и эффективность. В настоящем разделе будут рассмотрены некоторые методы оптимизации надстроек.

Скорость выполнения кода

Если попросить десять программистов провести автоматизацию определенной задачи, то вполне вероятно, что в результате будет получено десять различных решений. Чаще всего производительность всех этих решений будет находиться на одинаковом уровне.

Ниже приводятся советы, которые можно использовать для обеспечения максимального быстродействия кода.

- При записи данных на рабочий лист установите свойство `Application.ScreenUpdating` в значение `False`.
- Объявите тип данных всех применяемых переменных, а в завершенном коде любой ценой избегайте использования переменных типов. Используйте оператор `Option Explicit` в начале каждого модуля, чтобы сделать объявление типа переменной обязательным.
- Создайте переменные, чтобы избежать длинных ссылок на объекты. Например, если вы управляете объектом `Series` диаграммы, то необходимо создать переменную с помощью следующего кода.

  ```
  Dim S1 As Series
  Set S1 = ActiveWorkbook.Sheets(1).ChartObjects(1). _
      Chart.SeriesCollection(1)
  ```

- Объявите переменные объектов конкретного типа. Избегайте использования объявления `As Object`.

- Примените конструкцию `With-End With` для вызова нескольких методов и для изменения нескольких свойство одного объекта везде, где это возможно.
- Удалите избыточный код, особенно в записанных макросах.
- Если возможно, управляйте данными с помощью массивов VBA, а не диапазонов рабочих листов. Чтение и запись на рабочий лист занимает намного больше времени, чем обработка данных в памяти. Но это правило не всегда справедливо. Для получения наилучшего результата необходимо проверить быстродействие обоих вариантов.
- Избегайте связывания элементов управления пользовательских диалоговых окон с ячейками на рабочем листе, что может привести к пересчету рабочего листа при каждом изменении элемента управления в диалоговом окне.
- Перед созданием надстройки компилируйте используемый код. Это приведет к увеличению размера файла, но избавит от необходимости компилировать код перед каждым запуском процедур.

Размер файлов

Рабочие книги Excel (включая надстройки) всегда страдали от серьезной проблемы — огромного размера. Легко заметить, что размер файлов со временем увеличивается, даже если в рабочую книгу не добавляется содержимое. Это особенно справедливо, если удаляется большое количество кода, который заменяется на другой код. Внесение изменений в содержимое рабочих листов также приводит к увеличению размеров файлов.

Если необходимо создать надстройку (или просто рабочую книгу), занимающую минимальный объем, придется создать рабочую книгу заново.

1. Создайте резервную копию приложения и разместите ее в безопасном месте.
2. Активизируйте редактор VBE и экспортируйте все компоненты проекта, которые содержат код VBA (модули, модули кода, диалоговые окна `UserForm` и, возможно, `ThisWorkbook`, рабочие листы и модули диаграмм). Запишите информацию об именах файлов и их расположении.
3. Создайте новую рабочую книгу.
4. Скопируйте содержимое всех рабочих листов из первоначального приложения в рабочие листы новой рабочей книги. Особенно внимательно отнеситесь к именованным диапазонам в рабочей книге (их необходимо создать повторно).
5. Импортируйте компоненты, которые были экспортированы в п. 2.
6. Откомпилируйте код.
7. Если это необходимо, присоедините панели инструментов, которые использовались в первоначальной рабочей книге.
8. Сохраните новую рабочую книгу.
9. Проверьте новую рабочую книгу, чтобы удостовериться в том, что вами не потеряны компоненты.

Высока вероятность того, что созданный файл будет намного меньше, чем оригинал. Изменение размера зависит от ряда факторов, но, например, мне не удавалось уменьшить размер файла XLA более чем на 55%.

Особые проблемы, связанные с использованием надстроек

Надстройки являются достаточно мощным средством. Однако при их создании и использовании вас подстерегают некоторые сложности. Надстройки “страдают” собственным набором проблем, часть которых можно решить. В этом разделе будут рассмотрены вопросы, на которые необходимо обратить особое внимание при создании надстроек для массового распространения среди пользователей.

Правильная установка

В некоторых случаях требуется правильно установить надстройку (с помощью команды Сервис⇨Надстройки, а не Файл⇨Открыть). Этот раздел посвящен методам проверки правильности установки надстройки. Если надстройка установлена неправильно, то код VBA выполняет правильную установку (при необходимости добавляя надстройку в коллекцию AddIns) и использует окно сообщения для предоставления пользователю информации о выполненных действиях.

Листинг 21.1 содержит модуль кода для данного примера (предназначен для объекта ThisWorkbook). Такая техника основывается на возникновении события AddInInstall перед событием Open.

Листинг 21.1. Обеспечение правильной установки надстройки

```
Dim InstalledProperly As Boolean

Private Sub Workbook_AddinInstall()
    InstalledProperly = True
End Sub

Private Sub Workbook_Open()
    If Not ThisWorkbook.IsAddin Then Exit Sub

    If Not InstallProrerly Then
'       Добавление в коллекцию AddIns
       If Not InAddInCollection(ThisWorkbook) Then _
          AddIns.Add FileName:=ThisWorkbook.FullName

'      Установка надстройки
       AddInTitle = GetTitle(ThisWorkbook)
       Application.EnableEvents = False
       AddIns(AddInTitle).Installed = True
       Application.EnableEvents = True

'      Информирование пользователя
       Msg = ThisWorkbook.Name & _
         "установлена как надстройка"
       Msg = Msg & "Используйте команду Надстройки"
       MsgBox Msg, vbInformation, AddInTitle
    End If
End Sub
```

Если надстройка установлена правильно, то выполняется процедура Workbook_AddInInstall. Эта процедура устанавливает булеву переменную InstalledProperly в значение True. Если надстройка открыта с помощью команды Файл⇨Открыть, то процедура Workbook_AddInInstall не запускается, а переменная InstalledProperly имеет значение, принятое по умолчанию — False.

При выполнении процедуры `Workbook_Open` сначала проверяется, является ли рабочая книга надстройкой. Если это не так, то процедура завершается. Если рабочая книга является надстройкой, то процедура проверяет значение переменной `InstalledProperly`. В правильно установленной надстройке процедура завершает свое выполнение. В противном случае код вызывает определенную пользователем функцию, которая указывает на принадлежность надстройки к коллекции `AddIns`. Если это не так, то надстройка добавляется в коллекцию. После этого процедура завершается установкой файла в виде надстройки и созданием окна сообщения для пользователя. В целом результат использования этого кода заключается в корректной установке надстройки даже в случае выполнения команды Файл⇨Открыть. Кроме того, при выборе команды Файл⇨Открыть пользователь получает краткий урок по использованию надстроек.

Эта надстройка доступна на прилагаемом к книге компакт-диске. Попробуйте открыть ее с помощью команды Файл⇨Открыть, а затем — Сервис⇨Надстройки (правильный вариант установки).

Ссылки на другие файлы

Если надстройка использует другие файлы, то необходимо быть особенно внимательным при распространении приложения. Нельзя ничего предположить о файловой структуре системы, в которой пользователь запускает приложение. Самый простой метод — потребовать от пользователя расположить все файлы приложения в одной папке. Затем используйте свойство `Path` рабочей книги приложения, чтобы указать ссылки на необходимые файлы.

Например, если приложение имеет собственную справочную систему, то обязательно удостоверьтесь, что этот файл находится в той же папке, что и само приложение. Затем вы можете использовать представленную ниже процедуру для поиска файла справочных сведений.

```
Sub GetHelp()
    Dim Path As String
    Path = ThisWorkbook.Path
    Application.Help Path & "\USER.CHM"
End Sub
```

Если приложение использует функции API стандартных библиотек Windows, то можно ожидать, что Windows найдет все необходимые файлы. Однако если используется собственная библиотека DLL, то лучшим выходом будет сохранение этой библиотеки в папке `Windows\System` (которая может называться и по-другому). Для определения пути к папке `System` воспользуйтесь функцией Windows API `GetSystemDirectory`.

Указание правильной версии Excel

Если надстройка использует определенные возможности, которые уникальны в Excel 2003, то необходимо предупредить пользователей, решивших открыть надстройку в более ранней версии Excel. Следующий код выполняет эту нелегкую задачу.

```
Sub CheckVersion()
    If Val(Application.Version) < 11 Then
        MsgBox "Выполняется только в Excel 2003 и выше"
        ThisWorkbook.Close
    End If
End Sub
```

Свойство `Version` объекта `Application` возвращает строку. Например, свойство может возвратить значение 11.0a. Эта процедура использует функцию `Val`, которая игнорирует все элементы после первой буквы.

Дополнительная информация о совместимости приводится в главе 26.

Часть VI

Разработка приложений

В этой части...

Глава 22

Создание собственных панелей инструментов

В ЭТОЙ ГЛАВЕ...

Панели инструментов являются важным элементом пользовательского интерфейса. Их можно найти практически во всех современных продуктах. В этой главе рассмотрены следующие вопросы.

- Обзор командных панелей, к которым также относятся панели инструментов.
- Отслеживание панелей инструментов в Excel.
- Ручная настройка панелей инструментов.
- Примеры, демонстрирующие использование VBA для управления панелями инструментов.

В Excel не существует недостатка в панелях инструментов. Данная программа поставляется с более чем четырьмя десятками встроенных панелей инструментов. Кроме того, вы имеете возможность создавать новые панели инструментов и модифицировать существующие (и вручную, и с помощью кода VBA). В настоящей главе рассмотрены вопросы создания и модификации панелей инструментов.

О командных панелях

Начиная с Excel 97, Microsoft использует совершенно иной способ управления панелями инструментов по сравнению с предыдущими версиями. С технической точки зрения, панель инструментов известна как объект `CommandBar`. На самом деле тот объект, который обычно называется панелью инструментов, является частью командной панели.

- *Панель инструментов* — это плавающая панель с одним или несколькими элементами управления, которые поддерживают щелчки мышью. В данной главе основное внимание уделяется именно такому типу командных панелей.
- *Строка меню.* К встроенным строкам меню относятся Worksheet Menu Bar (Строка меню листа) и Chart Menu Bar (Строка меню диаграммы) (дополнительная информация о них приведена в главе 23).
- *Контекстное меню* — это меню, которое появляется при щелчке на объекте правой кнопкой мыши (дополнительная информация также содержится в главе 23).

ПЕРЕКРЕСТНАЯ ССЫЛКА

По причине того, что строка меню является командной панелью, практически вся информация этой главы справедлива и по отношению к первой. В главе 23 рассматриваются особенности управления пользовательскими меню.

Управление панелями инструментов

В представленном далее списке указываются изменения, которые допускается вносить в панели инструментов Excel.

- *Удаление элементов управления из встроенных панелей инструментов.* Вы можете избавиться от элементов управления, которые никогда не используются, что позволяет освободить несколько пикселей полезного места на экране.
- *Добавление элементов управления на встроенные панели инструментов.* На любую панель инструментов можно добавить любое количество элементов управления. Эти элементы управления могут быть кнопками, определенными пользователем, а также кнопками других панелей инструментов. Кроме того, на панель инструментов можно добавлять инструменты, которые встроены в Excel.
- *Создание новых панелей инструментов.* Вы вправе создать любое количество новых панелей инструментов. Элементы управления на них могут иметь любое происхождение.
- *Изменение функциональности элементов управления на встроенных панелях инструментов.* Данная задача выполняется путем добавления собственного макроса к элементу управления.
- *Изменение изображения, которое отображается на элементах управления.* Excel содержит ограниченный, но функционально насыщенный редактор кнопок панелей инструментов. Для изменения изображений существует несколько дополнительных способов.

Перечисленные выше операции можно выполнять вручную с помощью диалогового окна Настройка. Это диалоговое окно отображается при выборе нескольких команд Excel: Вид⇨Панели инструментов⇨Настройка, Сервис⇨Настройка. Вы также можете щелкнуть правой кнопкой мыши на любой панели инструментов и выбрать команду Настройка. Помимо этого, можно изменять панели инструментов с помощью кода VBA.



Не бойтесь экспериментировать с панелями инструментов. Если привести встроенную панель инструментов в нерабочее состояние, то всегда можно восстановить ее в прежнее состояние по умолчанию. Достаточно открыть диалоговое окно Настройка и перейти на вкладку Панели инструментов. После этого выберите панель инструментов в списке и щелкните на кнопке Сброс.

Как Excel обрабатывает панели инструментов

Перед началом работы с собственными панелями инструментов важно понимать общие принципы управления панелями инструментов. Представленная далее информация удивит не одного разработчика.

Хранение панелей инструментов

Панели инструментов могут быть присоединены к рабочим листам (XLS) или к файлам надстроек (XLA), что упрощает распространение собственных панелей инструментов вместе с приложениями (дополнительная информация приводится в разделе “Распространение панелей инструментов” далее в этой главе). К рабочей книге можно присоединить любое количество панелей инструментов. Когда пользователь открывает файл, присоединенная панель инструментов отображается автоматически.

Исключение составляет случай, когда в программе существует панель инструментов с таким же именем. В результате новая панель инструментов не будет заменять старую.

Excel хранит информацию о панелях инструментов в файле XLB, который находится в Windows XP по следующему адресу.

```
\Document and Settings\<имя пользователя>\Application Data\Microsoft\Excel
```

> Если вы обновили Excel до версии 2003 от ранней версии программы, то сведения из старого XLB-файла не переносятся в Excel 2003. Другими словами, вам придется проводить повторную настройку панелей инструментов.

Файлы XLB являются довольно важными. Предположим, ваш коллега использует рабочую книгу Excel, которая содержит пользовательскую панель инструментов. При открытии рабочей книги эта панель инструментов будет отображена на экране. После изучения документа принимается решение о том, что данная рабочая книга не нужна. Однако при выходе из Excel пользовательская панель инструментов будет сохранена в файле XLB. Если внести в панель инструментов любое изменение (от редактирования встроенной панели инструментов до создания новой панели инструментов), то при выходе из Excel файл XLB будет сохранен повторно. По причине того, что файл XLB загружается в память при запуске Excel, длительность запуска и выхода из Excel будет увеличиваться по мере увеличения размера файла XLB. Кроме того, панели инструментов занимают память и системные ресурсы. Таким образом, лучшим выходом будет отключение панелей инструментов, которые никогда не используются. Для этого необходимо воспользоваться командой Вид⇨Панели инструментов⇨Настройка.

> Частая причина сбоев — это невозможность загрузки поврежденного файла XLB. Если вы видите, что Excel не может загрузиться, то попробуйте переименовать XLB-файл.

Когда панели инструментов работают некорректно

Подход Excel к хранению панелей инструментов может привести к возникновению проблем. Предположим, что вами разработано приложение, которое использует дополнительную панель инструментов и эта панель инструментов присоединена к рабочей книге приложения. Когда конечный пользователь впервые открывает рабочую книгу приложения, панель инструментов отображается на экране. При закрытии Excel панель инструментов сохраняется в файле XLB в компьютере пользователя. Если пользователь каким-либо образом изменит панель инструментов (например, случайно удалит кнопку), то в следующий раз при запуске приложения панель инструментов в исходном виде отображена не будет. Вместо этого вы увидите модифицированную версию пользовательской панели инструментов, где отсутствует удаленная кнопка. Другими словами, панель инструментов со всеми необходимыми элементами управления, подключенная к рабочей книге, отображаться не будет, так как в окне программы открыта другая панель инструментов, которая имеет такое же имя. В большинстве случаев такое поведение не является приемлемым.

К счастью, можно создать код VBA, который будет препятствовать развитию этого сценария. Основной смысл заключается в том, чтобы не допустить добавления неправильной панели инструментов в коллекцию панелей инструментов. Такого результата можно добиться при создании панели инструментов “на лету” в момент открытия приложения и удаления панели инструментов перед завершением работы. При этом панель инструментов никогда не будет сохранена в файле XLB в компьютере пользователя. Может показаться, что создание панели инструментов ”на лету” — это чрезвычайно медленный процесс. Далее в этой главе показано, что создание панелей инструментов с помощью кода VBA выполняется довольно быстро.

Ручное управление панелями инструментов и кнопками

Excel содержит простые команды создания новых панелей инструментов и модификации существующих. При управлении панелями инструментов можно и не пользоваться программными средствами VBA, так как любые изменения в панели инструментов можно вносить вручную.

Важно понимать, что все изменения, которые вносятся в панели инструментов (как встроенные, так и пользовательские), являются "постоянными". Другими словами, изменения остаются в силе даже после перезапуска Excel, кроме того, они не связаны с определенной рабочей книгой. Чтобы вернуть панель инструментов в первоначальное состояние, ее необходимо сбросить (восстановить).

О режиме модификации командных панелей

Для того чтобы вручную внести изменения в панель инструментов (или меню), необходимо перевести Excel в *режим модификации командных панелей*. Для перевода Excel в этот режим можно воспользоваться одной из следующих команд.

- Вид⇨Панели инструментов⇨Настройка.
- Сервис⇨Настройка.
- Щелкните правой кнопкой мыши на панели инструментов или меню и выберите команду Настройка из контекстного меню.

Когда Excel переходит в режим модификации командных панелей, отображается диалоговое окно Настройка. В этом диалоговом окне можно вносить любые изменения в меню и панели инструментов. При щелчке правой кнопкой на меню и панелях инструментов отображается удобное контекстное меню (рис. 22.1). После применения проведенных изменений щелкните на кнопке Закрыть в диалоговом окне Настройка.

- Диалоговое окно Настройка содержит три вкладки.
- Панели инструментов. Находятся все доступные панели инструментов, включая пользовательские. В этот список входят две строки меню: строка меню листа и строка меню диаграммы, а также пользовательские строки меню.
- Команды. Перечисляет все доступные команды, разбитые по категориям. Эта вкладка может использоваться для добавления новых элементов на панель инструментов или панель меню.
- Параметры. Позволяет указывать различные параметры, которые относятся к панелям инструментов и меню. В число этих параметров входят размеры значков, состояние экранных подсказок, а также способ анимации эффектов меню.

Вкладка Параметры диалогового окна Настройка содержит флажок опции Всегда показывать полные меню. Рекомендуется установить этот флажок — если он сброшен, отображаются сокращенные меню. Разработчики Microsoft надеялись, что данный параметр не позволит начинающим пользователям запутаться в загроможденных опциями меню. Однако оказалось, что использование этого параметра привело к обратному результату.

В следующем разделе кратко описаны некоторые методы изменения панелей инструментов, которые можно вручную выполнять в диалоговом окне Настройка.

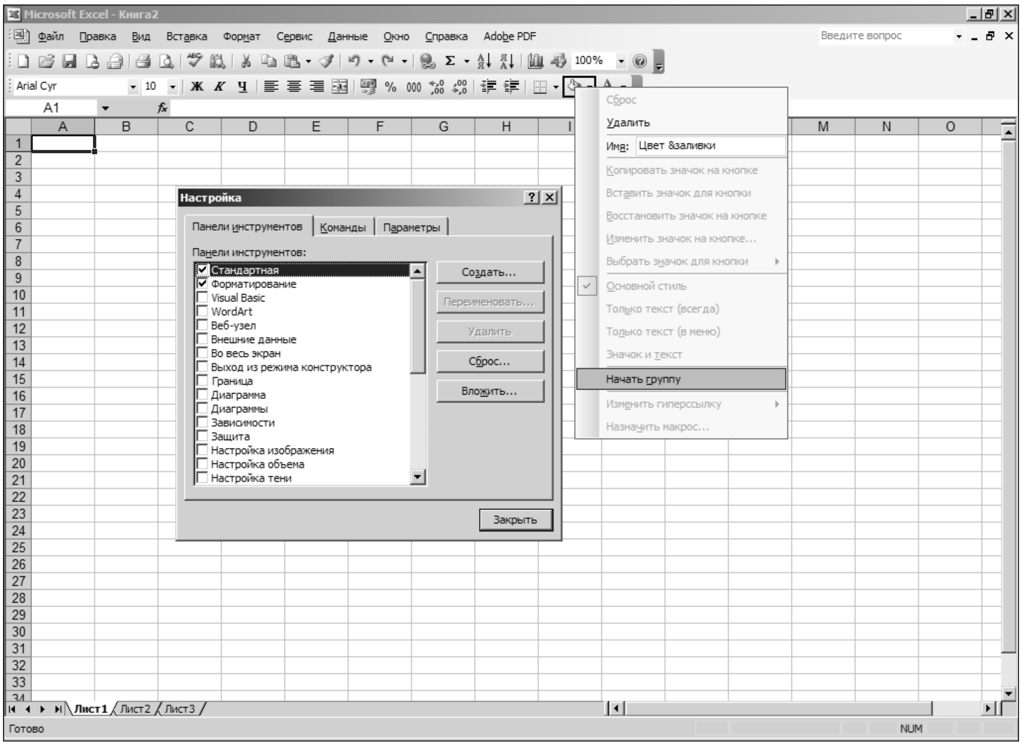


Рис. 22.1. В режиме модификации командных панелей допускается вносить изменения во все панели инструментов и меню

СКРЫТИЕ И ОТОБРАЖЕНИЕ ПАНЕЛЕЙ ИНСТРУМЕНТОВ

Вкладка Панели инструментов диалогового окна Настройка представляет имя каждой панели инструментов (как встроенной, так и пользовательской). Установите флажок напротив имени панели инструментов для того, чтобы отобразить эту панель на экране. Сброс флажка напротив имени приводит к скрытию панели инструментов. Изменения вступают в силу немедленно.

СОЗДАНИЕ НОВОЙ ПАНЕЛИ ИНСТРУМЕНТОВ

Щелкните на кнопке Создать и введите имя новой панели инструментов в диалоговом окне Создание панели инструментов. Excel создаст и отобразит пустую панель инструментов. На новую панель можно добавить необходимые кнопки (или команды меню).

На рис. 22.2 показана пользовательская панель инструментов, которая создана вручную. Эта панель инструментов называется Пользовательская панель и содержит наиболее часто используемые инструменты форматирования. Обратите внимание, что данная панель инструментов, кроме обычных кнопок, содержит меню раскрывающегося списка.

ПЕРЕИМЕНОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОЙ ПАНЕЛИ ИНСТРУМЕНТОВ

Выберите панель инструментов в списке и щелкните на кнопке Переименовать. Введите новое имя в диалоговом окне Переименование панели. Встроенную панель инструментов переименовать невозможно.

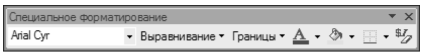


Рис. 22.2. Пользовательская панель инструментов, которая содержит инструменты форматирования

УДАЛЕНИЕ ПОЛЬЗОВАТЕЛЬСКОЙ ПАНЕЛИ ИНСТРУМЕНТОВ

Выберите панель инструментов в списке и щелкните на кнопке Удалить. Удалить встроенную панель инструментов невозможно.

СБРОС ВСТРОЕННОЙ ПАНЕЛИ ИНСТРУМЕНТОВ

Выберите встроенную панель инструментов из списка и щелкните на кнопке Сброс. Панель инструментов будет восстановлена в первоначальное состояние. Если на панель инструментов ранее были добавлены дополнительные инструменты, то они удаляются. Если удалены все инструменты, принятые по умолчанию, то они будут заново размещены на панели инструментов. Кнопка Сброс недоступна, пока выделена пользовательская панель инструментов.

ПЕРЕМЕЩЕНИЕ И КОПИРОВАНИЕ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ

Когда Excel находится в режиме модификации командных панелей (т.е. открыто диалоговое окно Настройка), можно копировать элементы управления между всеми видимыми панелями инструментов. Для того чтобы переместить элемент управления, перетащите его на новое место. Это может быть текущая или другая панель инструментов. Чтобы скопировать элемент управления, необходимо удерживать клавишу <Ctrl> в процессе перетаскивания элемента управления. Кроме того, элементы управления можно копировать в пределах одной панели инструментов.

ВСТАВКА НОВОГО ЭЛЕМЕНТА УПРАВЛЕНИЯ

Чтобы добавить новый элемент управления на панель инструментов, воспользуйтесь вкладкой Команды диалогового окна Настройка (рис. 22.3).

На этой вкладке элементы управления организованы в 17 категорий. При выборе категории элементы управления текущей категории отображаются справа. Чтобы узнать о выполняемых определенным элементом управления операциях, выберите его и щелкните на кнопке Описание. Чтобы добавить элемент управления на панель инструментов, выберите его на вкладке Команды и перетащите на панель инструментов.

ДОБАВЛЕНИЕ НА ПАНЕЛЬ ИНСТРУМЕНТОВ КНОПКИ ДЛЯ МАКРОСА

Вы можете создать на панели управления новую кнопку, к которой будет присоединен макрос. Для этого активизируйте вкладку Команды диалогового окна Настройка и выберите Макросы из списка Категории. Перетащите команду Настраиваемая кнопка на панель инструментов (по умолчанию на этой кнопке изображена улыбающаяся рожица).

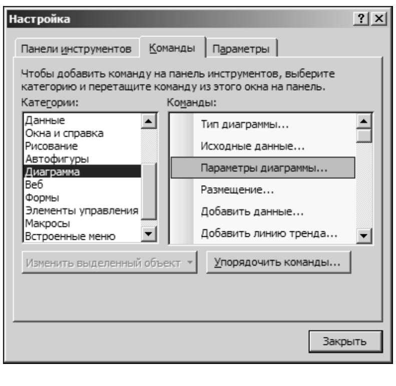


Рис. 22.3. Вкладка Команды *содержит список всех доступных встроенных элементов управления*

После добавления кнопки щелкните на ней правой кнопкой мыши и выберите необходимый параметр из появившегося контекстного меню, показанного на рис. 22.4. Измените название кнопки, укажите необходимый макрос и измените изображение.

НА ЗАМЕТКУ: Выбор команды Выбрать значок для кнопки из контекстного меню приводит к отображению меню из 42 изображений. Это небольшая часть библиотеки изображений, которые можно использовать. Дополнительная информация приведена в разделе "Изменение изображения на кнопках панели инструментов".

Распространение панелей инструментов

В этом разделе описаны методы распространения панелей инструментов в среде пользователей. Будут рассмотрены вопросы, с которыми зачастую сталкиваются при работе, но которые непросто устранить.

ДОБАВЛЕНИЕ ПАНЕЛИ ИНСТРУМЕНТОВ В РАБОЧУЮ КНИГУ

Для того чтобы сохранить панель инструментов в файле рабочей книги, выберите команду Вид⇨Панели инструментов⇨Настройка, и вы увидите диалоговое окно Настройка. Щелкните на кнопке Вложить, чтобы открыть диалоговое окно Управление панелями инструментов, которое показано на рис. 22.5. Это диалоговое окно включает список всех пользовательских панелей инструментов, находящихся в коллекции `Toolbars` (см. список слева). Панели инструментов, которые уже хранятся в рабочей книге, отображаются в списке справа.

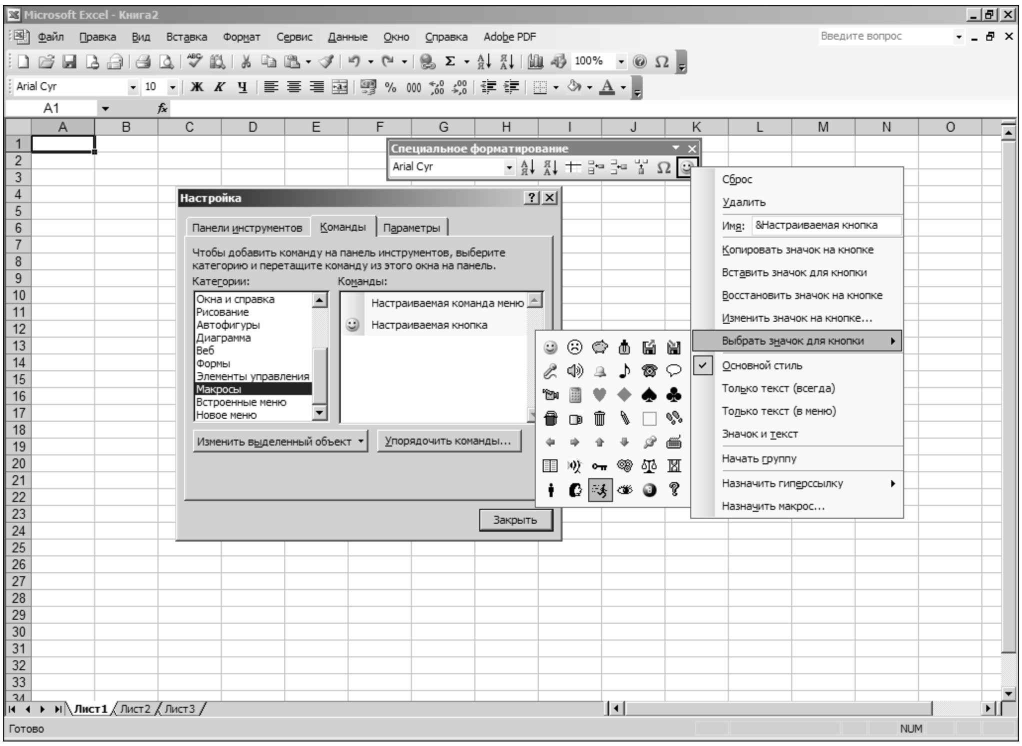


Рис. 22.4. Модификация кнопки на панели инструментов

Чтобы присоединить панель инструментов, выберите ее и щелкните на кнопке Копировать. Когда панель инструментов выделена в правом списке, кнопка Копировать преобразуется в Удалить. На этой кнопке можно щелкнуть для удаления панели инструментов из рабочей книги.

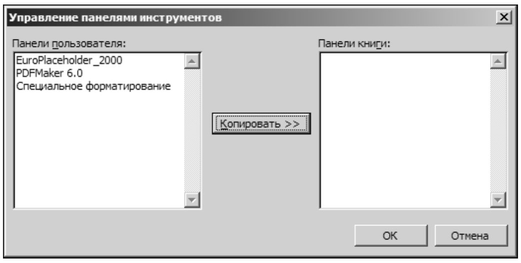


Рис. 22.5. Диалоговое окно Управление панелями инструментов

Странно, но не существует способа добавлять или удалять панели инструментов из рабочих книг с помощью кода VBA. Эти операции должны выполняться вручную.

Копия панели инструментов в рабочей книге всегда отображает содержимое панели инструментов в момент присоединения. Если в панель инструментов внести изменения после присоединения, то измененная версия не будет автоматически сохранена в рабочей книге. Для присоединения новой версии панели инструментов, следует вручную удалить старую версию из рабочей книги.

Панель инструментов, присоединенная к рабочей книге, автоматически отображается, когда пользователь открывает рабочую книгу, кроме случаев, если в рабочем пространстве уже присутствует панель инструментов с таким же именем. Дополнительная информация приводилась в разделе “Как Excel обрабатывает панели инструментов” ранее в этой главе.

РАСПРОСТРАНЕНИЕ ПАНЕЛЕЙ ИНСТРУМЕНТОВ С ПОМОЩЬЮ НАДСТРОЕК

Как отмечалось в главе 21, распространение надстройки часто является предпочтительным методом передачи приложения конечным пользователям. Надстройка может содержать одну или несколько дополнительных панелей инструментов. Но при этом необходимо помнить о потенциальной проблеме.

Приведем типичный сценарий: создается приложение, в котором применяется пользовательская панель инструментов. Кнопки на этой панели инструментов запускают процедуры VBA в рабочей книге приложения. Панель инструментов присоединяется к рабочей книге приложения, и рабочая книга сохраняется. После этого на основе рабочей книги создается надстройка. Версия XLS этого приложения закрывается, надстройка устанавливается, а при щелчке на кнопке панели управления надстройки *открывается рабочая книга из файла XLS!*

Предполагалось, конечно, что кнопки панели инструментов будут запускать процедуры из надстройки, а не из файла XLS, который служил основой для надстройки. Но при добавлении панели инструментов к рабочей книге панель инструментов добавляется в своем текущем виде. В этом виде на панели инструментов используются ссылки на макросы в рабочей книге файла XLS. Следовательно, щелчок на кнопке

приводит к открытию файла XLS, что необходимо для запуска хранящегося в нем макроса. Можно вручную (или с помощью кода VBA) изменить свойство `OnAction` каждой кнопки панели инструментов, чтобы она ссылалась на код надстройки, а не на код, хранящийся в файле XLS. Однако рекомендуется создать код, который будет создавать панель инструментов "на лету", пока происходит открытие надстройки. Решение данной задачи приводится далее в этой главе.

Управление коллекцией CommandBars

Коллекция `CommandBars`, которая содержится в объекте `Application`, представляет все объекты `CommandBar`. Каждый объект `CommandBar` содержит коллекцию объектов `Control`. Все они имеют свойства и методы, которые можно использовать в процедурах языка VBA.

В данном разделе приведены основные сведения о создании кода управления панелями инструментов (как всегда, понимание объектной модели значительно упростит вам жизнь).

Командными панелями Excel (в число которых входят и панели управления) можно управлять с помощью объектов, входящих в состав коллекции `CommandBars`. Эта коллекция содержит следующие элементы.

- 63 встроенные панели инструментов Excel 2003.
- Все пользовательские панели инструментов.
- Встроенная строка меню листа. Эта строка отображается, когда активен рабочий лист.
- Встроенная строка меню диаграммы. Эта строка отображается, когда активен лист диаграммы.
- Все создаваемые строки меню.
- Все 61 контекстное меню, которые интегрированы в Excel 2003.

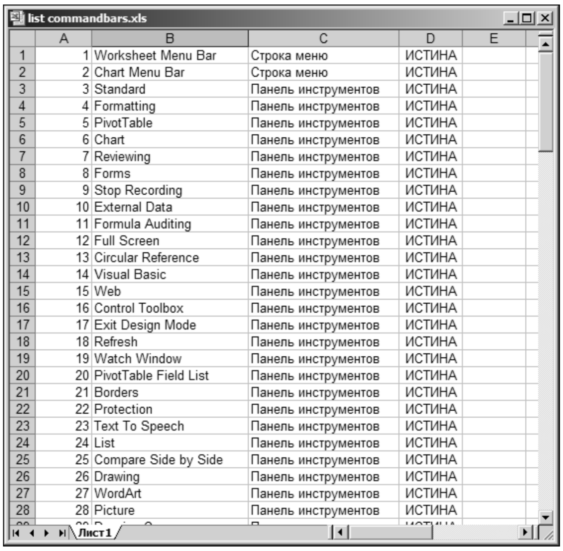
Типы командных панелей

Как отмечалось в самом начале этой главы, существует три типа командных панелей, каждый из которых отличается значением свойства `Type`. Возможные значения этого свойства коллекции `CommandBars` перечислены в следующей таблице. VBA имеет встроенные константы, которые используются для указания типа командной панели.

Тип	Описание	Константа
0	Панель инструментов	`msoBarTypeNormal`
1	Строка меню	`msoBarTypeMenuBar`
2	Контекстное меню	`msoBarTypePopUp`

Просмотр списка объектов CommandBar

Если вы решили получить информацию о содержимом коллекции `CommandBars`, то следующая процедура окажется весьма полезной. Ее запуск приведет к созданию списка (рис. 22.6), который содержит все объекты `CommandBar`, входящие в коллекцию `CommandBars`. Для Excel 2003 этот список состоит из 126 встроенных командных панелей и всех командных панелей, определенных пользователем. Для каждой командной панели процедура указывает значения свойств `Index`, `Type` и `Name` (свойство `Type` отображается в виде *Панель инструментов*, *Строка меню* или *Контекстное меню*). Кроме того, указывается, какая из командных панелей является встроенной.



	A	B	C	D	E
1	1	Worksheet Menu Bar	Строка меню	ИСТИНА	
2	2	Chart Menu Bar	Строка меню	ИСТИНА	
3	3	Standard	Панель инструментов	ИСТИНА	
4	4	Formatting	Панель инструментов	ИСТИНА	
5	5	PivotTable	Панель инструментов	ИСТИНА	
6	6	Chart	Панель инструментов	ИСТИНА	
7	7	Reviewing	Панель инструментов	ИСТИНА	
8	8	Forms	Панель инструментов	ИСТИНА	
9	9	Stop Recording	Панель инструментов	ИСТИНА	
10	10	External Data	Панель инструментов	ИСТИНА	
11	11	Formula Auditing	Панель инструментов	ИСТИНА	
12	12	Full Screen	Панель инструментов	ИСТИНА	
13	13	Circular Reference	Панель инструментов	ИСТИНА	
14	14	Visual Basic	Панель инструментов	ИСТИНА	
15	15	Web	Панель инструментов	ИСТИНА	
16	16	Control Toolbox	Панель инструментов	ИСТИНА	
17	17	Exit Design Mode	Панель инструментов	ИСТИНА	
18	18	Refresh	Панель инструментов	ИСТИНА	
19	19	Watch Window	Панель инструментов	ИСТИНА	
20	20	PivotTable Field List	Панель инструментов	ИСТИНА	
21	21	Borders	Панель инструментов	ИСТИНА	
22	22	Protection	Панель инструментов	ИСТИНА	
23	23	Text To Speech	Панель инструментов	ИСТИНА	
24	24	List	Панель инструментов	ИСТИНА	
25	25	Compare Side by Side	Панель инструментов	ИСТИНА	
26	26	Drawing	Панель инструментов	ИСТИНА	
27	27	WordArt	Панель инструментов	ИСТИНА	
28	28	Picture	Панель инструментов	ИСТИНА	

Рис. 22.6. Код VBA создает такой список объектов `CommandBar`

```
Sub ShowCommandBarNames()
    Dim Row As Integer
    Dim cbar As CommandBar
    Cells.Clear
    Row = 1
    For Each cbar In CommandBars
        Cells(Row, 1) = cbar.Index
        Cells(Row, 2) = cbar.Name
        Select Case cbar.Type
            Case msoBarTypeNormal
                Cells(Row, 3) = "Панель инструментов"
            Case msoBarTypeMenuBar
                Cells(Row, 3) = "Строка меню"
            Case msoBarTypePopUp
                Cells(Row, 3) = "Контекстное меню"
        End Select
        Cells(Row, 4) = cbar.BuiltIn
        Row = Row + 1
    Next cbar
End Sub
```

НА ЗАМЕТКУ

При работе с панелями инструментов можно включить запись макросов, чтобы получить представление о том, что происходит, в терминах VBA. Большинство (но не все) действий, предпринимаемых для изменения панелей инструментов, приводят к генерации кода VBA. Рассматривая записанный код, легко понять, как устроена объектная модель панелей инструментов. Вскоре вы убедитесь, что объектная модель панелей инструментов достаточно проста и легка для понимания.

Создание командной панели

В VBA командная панель создается с помощью метода `Add` коллекции `CommandBars`. Представленный далее оператор создает новую панель инструментов, которая имеет имя, принятое по умолчанию (`Custom 1`).

```
CommandBars.Add
```

Созданная панель инструментов не содержит элементов управления, кроме того, пользователь не видит ее (свойство `Visible` установлено в значение `False`).

Зачастую при создании панели инструментов возникает необходимость установить некоторые ее свойства. Следующий пример демонстрирует один из способов выполнить данную задачу.

```
Sub CreateToolbar()
    Dim TBar As CommandBar
    Set TBar = CommandBars.Add
    With TBar
        .Name = "МояПанель"
        .Top = 0
        .Left = 0
        .Visible = True
    End With
End Sub
```

Процедура `CreateToolbar` использует метод `Add` коллекции `CommandBars` для добавления новой панели инструментов и создает переменную `Tbar`, которая представляет созданную панель инструментов. Следующие инструкции назначают панели инструментов имя, располагают ее в верхнем левом углу экрана и делают видимой. Свойства `Top` и `Left` указывают расположение панели инструментов, задавая координаты на экране, а не в окне программы Excel.

При получении доступа к коллекции `CommandBars` из модуля кода диалогового окна `UserForm` или объектов `ЭтаКнига`, `Лист`, `Диаграмма` необходимо добавлять к ссылкам объект `Application`.

```
Application.CommandBars.Add
```

Если код находится в модуле кода VBA общего назначения, то задавать ссылки не обязательно.

Ссылки на командные панели

На определенный объект `CommandBar` можно ссылаться с помощью значения свойства `Index` или свойства `Name`. Например, панель инструментов Standard (Стандартная) имеет свойство `Index`, установленное в значение 3. Это позволяет ссылаться на эту панель инструментов с помощью таких операторов.

```
CommandBars(3)
CommandBars("Standard")
```

Если для обращения к панели инструментов используется имя, то помните о регистре символов. Другими словами, можно одновременно использовать для разных панелей инструментов такие имена, как `Standard`, `standard` и `STANDARD`.

В разных версиях Excel панели инструментов имеют различные значения свойства `Index` (нумерация индексов панелей не сохраняется). Например, в Excel 2002 панель инструментов 3-D Settings (Настройка объема) имеет свойство `Index`, установленное в значение 58. А в Excel 2000 значение 58 свойства `Index` соответствует панели инструментов WordArt. Если приложение должно работать с различными версиями Excel, то необходимо использовать свойство `Name`, а не `Index`.

Удаление командных панелей

Для того чтобы удалить панель инструментов, определенную пользователем, необходимо сослаться на объект с помощью индекса (если он известен) или имени. Следующий оператор удаляет панель инструментов, которая называется `МояПанель`.

```
CommandBars("МояПанель").Delete
```

Если панель инструментов не существует, то этот оператор приведет к появлению сообщения об ошибке. Для того чтобы избежать появления сообщения об ошибке при удалении панели инструментов, которая может не существовать, достаточно проигнорировать эту ошибку. Следующий код удаляет панель инструментов `МояПанель`, если такая панель существует. Если она не существует, то сообщение об ошибке на экране не отображается.

```
On Error Resume Next
CommandBars("МояПанель").Delete
On Error GoTo 0
```

Еще один подход заключается в создании собственной функции, которая определяет наличие определенной панели инструментов в коллекции `CommandBars`. Представленная далее функция принимает в качестве параметра имя объекта `CommandBar` (который потенциально существует) и возвращает значение `True`, если эта командная панель присутствует. Такая функция циклически просматривает элементы коллекции `CommandBars` и завершает свою работу после того, как будет найдено имя, указанное в виде параметра.

```
Function CommandBarExists(n) As Boolean
    Dim cb As CommandBar
    For Each cb In CommandBars
        If UCase(cb.Name) = UCase(n) Then
            CommandBarExists = True
            Exit Function
        End If
    Next cb
    CommandBarExists = False
End Function
```

Свойства командных панелей

Ниже приведены наиболее полезные свойства объекта `CommandBar`.

- `BuiltIn` — предназначено только для чтения. Имеет значение `True`, если объект является одной из встроенных в Excel командных панелей.
- `Left` — координаты на экране левого края командной панели (в пикселях).
- `Name` — отображаемое имя командной панели.
- `Position` — целое число, которое отображает позицию командной панели. Возможны следующие значения этого свойства.
    - `msoBarLeft` — командная панель прикреплена к левой границе окна.
    - `msoBarTop` — командная панель прикреплена к верхней границе окна.
    - `msoBarRight` — командная панель прикреплена к правой границе окна.
    - `msoBarBottom` — командная панель прикреплена к нижней границе окна.
    - `msoBarFloating` — командная панель ни к чему не прикреплена (плавающая).
    - `msoBarPopup` — командная панель представляет контекстное меню.

- `Protection` — целое число, которое указывает тип защиты командной панели. Это свойство может иметь следующие значения.
  - `msoBarNoProtection` — (значение по умолчанию) защита не применяется. Командная панель может изменяться пользователем.
  - `msoBarNoCustomize` — командная панель не может изменяться.
  - `msoBarNoResize` — не может изменяться размер командной панели.
  - `msoBarNoMove` — командная панель не может перемещаться.
  - `msoBarNoChangeVisible` — пользователь не может изменить состояние видимости этой командной панели.
  - `msoBarNoChangeDock` — пользователь не может прикрепить панель в другом положении.
  - `msoBarNoVerticalDock` — командная панель не прикрепляется к правому или левому краю окна.
  - `msoBarNoHorizontalDock` — командная панель не прикрепляется к верхнему или нижнему краю окна.
- `Top` — задает вертикальную позицию командной панели (в пикселях).
- `Type` — содержит целое число, которое представляет тип командной панели. Это свойство может иметь следующие значения.
  - `msoBarTypeNormal` — панель инструментов.
  - `msoBarTypeMenuBar` — строка меню.
  - `msoBarTypePopUp` — контекстное меню.
- `Visible` — свойство имеет значение `True` для отображаемых командных панелей.

Примеры кода VBA, которые приводятся в следующем разделе, демонстрируют использование некоторых свойств командных панелей.

ПОДСЧЕТ ПОЛЬЗОВАТЕЛЬСКИХ ПАНЕЛЕЙ ИНСТРУМЕНТОВ

Следующая функция возвращает количество панелей инструментов, которые определены пользователем. Эта функция в цикле перебирает элементы коллекции `CommandBars` и увеличивает значение счетчика на единицу каждый раз, когда элемент коллекции является панелью инструментов и его свойство `BuiltIn` не установлено в значение `True`.

```
Function CustomToolbars()
    Dim cb As CommandBar
    Dim Count As Integer
    Count = 0
    For Each cb In CommandBars
        If cb.Type = msoBarTypeNormal Then
            If Not cb.BuiltIn Then
                Count = Count + 1
            End If
        End If
    Next cb
    CustomToolbars = Count
End Function
```

ЗАПРЕТ ИЗМЕНЕНИЯ ПАНЕЛИ ИНСТРУМЕНТОВ

Свойство Protection объекта CommandBar предоставляет ряд возможностей для защиты панели инструментов. Приведенный ниже оператор устанавливает свойство Protection панели инструментов МояПанель.

```
CommandBars("МояПанель").Protection = msoBarNoCustomize
```

После выполнения этого оператора пользователь не сможет модифицировать панель инструментов.

Константы, которые используются в качестве значений свойства Protection, являются аддитивными, что означает одновременное суммирование различных степеней защиты. Например, следующий оператор запрещает модификацию и перемещение панели инструментов МояПанель.

```
Set cb = CommandBars("МояПанель")
cb.Protection = msoBarNoCustomize + msoBarNoMove
```

СОЗДАНИЕ "АВТОМАТИЧЕСКОЙ" ПАНЕЛИ ИНСТРУМЕНТОВ

Некоторые встроенные панели инструментов Excel обладают определенным "уровнем интеллекта" (как может показаться). Они появляются при возникновении конкретного объекта и исчезают при его скрытии. Например, панель инструментов Chart (Диаграммы) обычно отображается только тогда, когда ведется работа над диаграммой. Как только работа с диаграммой завершается, панель инструментов исчезает. Одно время Microsoft называла это поведение *чувствительностью панелей инструментов (toolbar autosensing)*. Но в последующих версиях этот термин не использовался. По причине того, что более удачное название отсутствует, в этой книге будет использоваться термин *чувствительность (autosensing)* для описания автоматического поведения панелей инструментов.

Чтобы отключить чувствительность определенной панели инструментов, достаточно закрыть ее при работе с таким объектом, для которого отображается панель инструментов. Чтобы повторно включить чувствительность, отобразите панель при работе с соответствующим объектом.

Зачастую возникает необходимость программирования чувствительности панелей в собственном приложении. Например, может понадобиться сделать панель инструментов видимой только тогда, когда активными становятся определенные рабочие листы или ячейки в диапазоне. Благодаря поддержке событий в программной модели Excel такое программирование осуществить достаточно просто.

Процедура, представленная в листинге 22.1, создает панель инструментов при открытии рабочей книги. Событие SelectionChange рабочей книги используется для определения принадлежности активной ячейки диапазону ToolbarRange. Если это так, то панель инструментов отображается на экране. В противном случае панель инструментов скрывается. Другими словами, панель инструментов видима только тогда, когда активная ячейка входит в состав определенного диапазона рабочего листа.

Процедура, называемая Workbook_Open, создает простую панель инструментов с именем Autosense. Четыре кнопки на панели инструментов, настроенные для вызова процедур, называются Button1, Button2, Button3 и Button4. Обратите внимание, что перед созданием панели инструментов программно удаляется одноименная панель инструментов (если такая существует).

Листинг 22.1. Пользовательская панель инструментов, отображаемая только при активности ячейки определенного диапазона

```
Sub CreateToolbar()
'   Создание демонстрационной панели инструментов AutoSense
    Dim AutoSense As CommandBar
    Dim Button As CommandBarButton
    Dim i As Integer

'   Удаление существующей одноименной панели инструментов
    On Error Resume Next
    CommandBars("AutoSense").Delete
    On Error GoTo 0

'   Создание панели инструментов
    Set AutoSense = CommandBars.Add
    For i = 1 To 4
        Set Button = AutoSense.Controls.Add(msoControlButton)
        With Button
            .OnAction = "Button" & i
            .FaceId = i + 37
        End With
    Next i
    AutoSense.Name = "AutoSense"
End Sub
```

Процедура обработки события SelectionChange (которая расположена в модуле кода рабочего листа Sheet1) выглядит следующим образом.

```
Private Sub Worksheet_SelectionChange(ByVal Target As _
 Excel.Range)
    If Union(Target, Range("ToolbarRange")).Address = _
      Range("ToolbarRange").Address Then
        CommandBars("AutoSense").Visible = True
    Else
        CommandBars("AutoSense").Visible = False
    End If
End Sub
```

Эта процедура проверяет адрес активной ячейки. Если такая ячейка находится в указанном диапазоне, который называется ToolbarRange, то свойство Visible панели инструментов AutoSense устанавливается в значение True. В противном случае значение свойства Visible имеет значение False.

Кроме того, рабочая книга содержит процедуру Workbook_BeforeClose, которая удаляет панель инструментов AutoSense при закрытии рабочей книги. Эта техника, конечно, может быть адаптирована для предоставления других видов чувствительности панелей инструментов.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Чтобы получить исчерпывающее описание всех типов событий, которые поддерживает Excel, необходимо обратиться к главе 19.

СКРЫТИЕ (С ПОСЛЕДУЮЩИМ ОТОБРАЖЕНИЕМ) ВСЕХ ПАНЕЛЕЙ ИНСТРУМЕНТОВ

Некоторые разработчики стремятся получить контроль над всеми функциями Excel в процессе работы приложения. Например, может возникнуть необходимость в скрытии всех панелей инструментов, строки состояния и панели формул. При этом после закрытия приложения все панели инструментов и элементы управления должны стать видимыми и вернуться в состояние, которое имели до загрузки приложения.

Пример, приведенный в этом разделе, демонстрирует способ скрытия всех панелей инструментов и последующего их отображения после закрытия приложения. Процедура `HideAllToolbars` вызывается из процедуры обработки события `Workbook_Open`. Процедура `RestoreToolbars` вызывается из процедуры обработки события `Workbook_BeforeClose`.

Код отслеживает панели инструментов, которые отображены на экране, сохраняя их имена в рабочем листе `TBSheet`. Когда рабочая книга закрывается, процедура `RestoreToolbars` считывает список из ячеек рабочего листа и отображает необходимые панели инструментов. Использование рабочего листа для хранения имен панелей инструментов является более безопасным способом хранения, чем применение для этих же целей массива VBA (который может легко изменить свои значения). Обе процедуры приводятся в листинге 22.2.

Листинг 22.2. Скрытие всех панелей инструментов с последующим их отображением

```
Sub HideAllToolbars()
    Dim TB As CommandBar
    Dim TBNum As Integer
    Dim TBSheet As Worksheet
    Set TBSheet = Sheets("TBSheet")
    Application.ScreenUpdating = False

'   Очистка листа
    TBSheet.Cells.Clear

'   Скрыть видимые панели инструментов и сохранить их названия

    TBNum = 0
    For Each TB In CommandBars
        If TB.Type = msoBarTypeNormal Then
            If TB.Visible Then
                TBNum = TBNum + 1
                TB.Visible = False
                TBSheet.Cells(TBNum, 1) = TB.Name
            End If
        End If
    Next TB
    Application.ScreenUpdating = True
End Sub

Sub RestoreToolbars()
    Dim TBSheet As Worksheet
    Dim cell As Range

    Set TBSheet = Sheets("TBSheet")
    Application.ScreenUpdating = False

'   Отобразить скрытые ранее панели инструментов
    On Error Resume Next
    For Each cell In TBSheet.Range("A:A") _
      .SpecialCells(xlCellTypeConstants)
        CommandBars(cell.Value).Visible = True
    Next cell
    Application.ScreenUpdating = True
End Sub
```

НА ЗАМЕТКУ

В некоторых случаях может оказаться, что скрытие видимой панели инструментов выполняется неправильно. Ведь панель инструментов с чувствительностью может продолжать появляться в соответствующем контексте. Одним из решений этой проблемы может служить установка свойства `Enabled` в значение `False` для всех панелей инструментов, которые не должны отображаться на экране.

Ссылка на элементы управления командной панели

Такой объект `CommandBar`, как панель инструментов, содержит объекты `CommandBarControl`. Эти объекты — это свойства `Controls` объекта `CommandBar`.

- `Button`. Объект кнопки (`CommandBarButton`).
- `ComboBox`. Объект списка (`CommandBarComboBox`).
- `Menu`. Объект меню (`CommandBarPopup`).

Приведенная ниже процедура `Test` отображает значение свойства `Caption` для первого объекта `Control`, который находится на панели инструментов Standard (Стандартная). Этот объект `CommandBar` имеет индекс 3.

```
Sub Test()
    MsgBox CommandBars(3).Controls(1).Caption
End Sub
```

При выполнении данной процедуры будет отображено окно сообщения, которое показано на рис. 22.7 (предполагается, что панель инструментов Standard (Стандартная) не изменялась).

Использование индексов для доступа к объектам панелей инструментов справедливо, независимо от того, установлен ли параметр Всегда показывать полные меню (этот флажок расположен на вкладке Параметры диалогового окна Настройка).

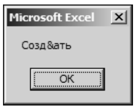


Рис. 22.7. Отображение свойства `Caption` элемента управления

Вместо номера индекса при создании ссылки на элемент управления, можно использовать значение свойства `Caption`. Представленная далее процедура приводит к получению такого же результата, что и предыдущая.

```
Sub Test2()
    MsgBox CommandBars("Standard").Controls("Создать").Caption
End Sub
```

ВНИМАНИЕ! Ссылка на элемент управления с помощью названия зависит от используемого языка пользовательского интерфейса. Таким образом, пример, приведенный выше, возможно, не будет работать в неанглийской версии Excel. Решением в данной ситуации может оказаться использование метода `FindControl`, который позволяет проводить поиск элементов управления на основе значения свойства `Id`. Применение данного метода рассматривается в главе 23.

Если отображается свойство `Caption` элемента управления, то можно заметить, что значение данного свойства содержит символ амперсанда (&). Буква, которая следует после этого символа, задает подчеркнутую горячую клавишу в отображаемом тексте (например, `Созд&ать`). При ссылке на элемент управления с использованием значения свойства `Caption` указывать символ амперсанда не требуется.

ПЕРЕКРЕСТНАЯ ССЫЛКА. В некоторых случаях объекты `Control` могут содержать другие объекты `Control`. Например, первый элемент управления на панели инструментов Drawing (Рисование) содержит другие элементы управления (этот пример демонстрирует способ добавления меню на панель инструментов). Концепция объектов `Control`, которые содержатся внутри других объектов `Control`, более подробно рассмотрена в главе 23, где освещается вопрос создания меню.

Перечисление элементов управления на командной панели

Следующая процедура отображает свойство `Caption` каждого объекта `Control`, содержащего внутри объекта `CommandBar`. В приведенном примере используется панель инструментов Standard (Стандартная).

```
Sub ShowControlCaptions()
    Dim Cbar as CommandBar
    Set CBar = CommandBars("Standard")
    Cells.Clear
    Row = 1
    For Each ctl In CBar.Controls
        Cells(Row, 1) = ctl.Caption
        Row = Row + 1
    Next ctl
End Sub
```

Результат выполнения процедуры `ShowControlCaptions` показан на рис. 22.8.

Перечисление элементов управления на всех панелях инструментов

Отображенная далее процедура просматривает все командные панели в коллекции. Если командная панель является панелью инструментов (т.е. ее свойство `Type` установлено в значение 1), то другой цикл отображает значение свойства `Caption` для каждой кнопки на панели инструментов.

Частичный результат работы процедуры `ShowAllToolbarControls` показан на рис. 22.9.

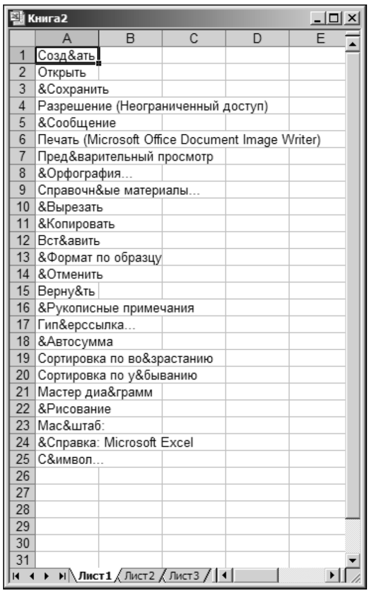


Рис. 22.8. Список подписей для каждого элемента управления, который находится на панели инструментов Standard (Стандартная)

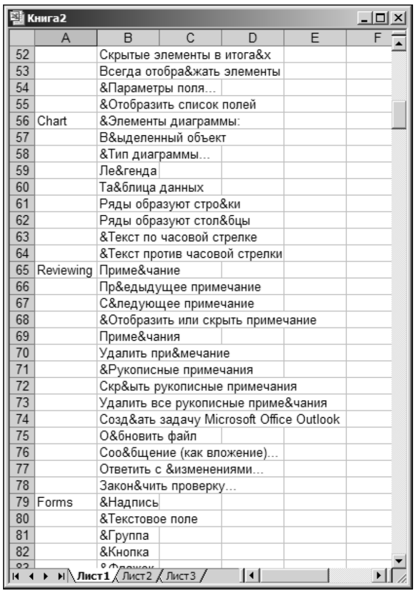


Рис. 22.9. Список названий всех элементов управления на всех панелях инструментов

```
Sub ShowAllToolbarControls()
    Dim row As Integer
    Dim Cbar As CommandBar
    Dim ctl As CommandBarControl

    Cells.Clear
    row = 1
    For Each Cbar In CommandBars
        If Cbar.Type = msoBarTypeNormal Then
            Cells(row, 1) = Cbar.Name
            For Each ctl In Cbar.Controls
                Cells(row, 2) = ctl.Caption
                row = row + 1
            Next ctl
        End If
    Next Cbar
End Sub
```

Добавление элементов управления на командную панель

Для того чтобы добавить элемент управления в объект CommandBar, можно воспользоваться методом Add коллекции Controls. Приведенный ниже оператор добавляет элемент управления на панель инструментов МояПанель. Свойство Type этого элемента управления установлено в значение, равное встроенной константе msoControlButton, что приводит к созданию стандартной кнопки.

```
CommandBars("МояПанель").Controls.Add _
     Type:=msoControlButton
```

Кнопка, добавленная на панель инструментов с помощью предыдущего оператора, не имеет функций. Щелчок на этой кнопке не приведет ни к какому результату. Как правило, б*о*льшая часть свойств элемента управления устанавливается в момент добавления элемента управления на панель инструментов. Следующий код добавляет новый элемент управления, назначает ему рисунок с помощью свойства FaceId, назначает макрос посредством свойства OnAction и указывает подпись.

```
Sub AddButton()
    Dim NewBtn As CommandBarButton
    Set NewBtn = CommandBars("МояПанель").Controls.Add _
      (Type:=msoControlButton)
    With NewBtn
       .FaceId = 300
       .OnAction = "МойМакрос"
       .Caption = "Экранная подсказка"
    End With
End Sub
```

Процедура AddButton создает переменную (NewBtn), которая представляет добавленный элемент управления. После этого конструкция With-With End устанавливает свойства объекта.

Того же результата можно добиться без использования переменной объекта, как показано в следующем примере.

```
Sub AddButton()
    With CommandBars("МояПанель").Controls.Add _
          (Type:=msoControlButton)
       .FaceId = 300
       .OnAction = "МойМакрос"
       .Caption = "Экранная подсказка"
    End With
End Sub
```

Удаление элемента управления из командной панели

Для того чтобы удалить элемент управления из объекта CommandBar, воспользуйтесь методом Delete коллекции Controls. Следующий оператор удаляет первый элемент управления на панели инструментов МояПанель.

```
CommandBars("МояПанель").Controls(1).Delete
```

Кроме того, можно определить элемент управления, сославшись на его название. Приведенный далее оператор удаляет элемент управления, который имеет название Сортировка.

```
CommandBars("МаяПанель").Controls("Сортировка").Delete
```

Свойства элементов управления командных панелей

Как отмечалось ранее, элементы управления командных панелей имеют набор свойств, которые определяют внешний вид и поведение элементов управления. Ниже приведен список свойств элементов управления командных панелей.

- BeginGroup — если это свойство установить в значение True, то перед элементом управления будет отображена разделительная полоса.
- BuiltIn — свойство имеет значение True, если элемент управления является одним из встроенных элементов управления Excel (предназначено только для чтения).
- Caption — значение этого свойства отображается в виде текста подписи элемента управления. Если элемент управления имеет только значок, то подпись будет отображена лишь после наведения на элемент управления указателя мыши.
- Enabled — если это свойство установлено в значение True, то на элементе управления можно щелкнуть.
- FaceId — число, которое обозначает изображение, представленное возле текста на элементе управления.
- Id — кодовый номер предопределенной команды Excel (предназначено только для чтения).
- OnAction — имя процедуры VBA, которая будет запускаться при щелчке на этом элементе управления.
- State — определяет состояние "нажатия" элемента управления. Это свойство поддерживается только элементами управления CommandBarButton.
- Style — определяет состояние отображения элемента управления с текстом и изображением. Это свойство поддерживается лишь элементами управления CommandBarButton и CommandBarComboBox.
- ToolTipText — текст, отображаемый при наведении пользователем указателя мыши на элемент управления.
- Type — целое число, которое определяет тип элемента управления.

УСТАНОВКА СВОЙСТВА STYLE ЭЛЕМЕНТА УПРАВЛЕНИЯ

Свойство Style элемента управления определяет его внешний вид (это свойство относится только к элементам управления CommandBarButton и CommandBarComboBox). Значение данного свойства обычно устанавливается с помощью встроенной константы.

Например, чтобы отобразить кнопку с текстом и изображением, установите свойство `Style` в значение `msoButtonIconAndCaption`. Ниже представлены допустимые значения свойства `Style` элементов управления `CommandBarButton`:

- `msoButtonAutomatic`;
- `msoButtonCaption`;
- `msoButtonIcon`;
- `msoButtonAndCaption`;
- `msoButtonIconAndCaptionBelow`;
- `msoButtonAndIconAndWrapCaption`;
- `msoButtonAndIconAndWrapCaptionBelow`;
- `msoButtonWrapCaption`.

Для элемента управления `CommandBarComboBox` допустимыми значениями являются `msoComboLabel` и `msoComboNormal`.

На рис. 22.10 показана панель инструментов, на которой находится семь элементов управления (командных кнопок). Каждый элемент управления имеет свой стиль.

Рабочая книга, которая создает такую панель инструментов, доступна на прилагаемом к книге компакт-диске.

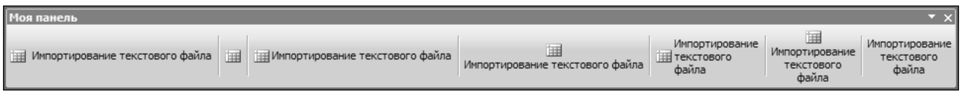


Рис. 22.10. Семь значений свойства `Style` для элементов управления (кнопки)

Текст, который отображается на элементе управления, представлен значением свойства `Caption` этого элемента управления. Отображаемый значок задается значением свойства `FaceId`.

ИЗМЕНЕНИЕ ЗНАЧКА НА КНОПКЕ

Когда Excel переходит в режим изменения командных панелей, можно щелкнуть правой кнопкой мыши на любой панели инструментов и выбрать команду **Выбрать значок для кнопки**. Это действие позволяет отобразить меню из 42 изображений, которые могут использоваться в качестве значков на элементах управления. Как правило, ни один из этих значков полностью не удовлетворяет запросы разработчика. Поэтому всегда можно указать значок с помощью кода VBA.

Изображение (если оно демонстрируется), связанное с элементом управления на панели инструментов, определяется значением свойства `FaceId`. Для того чтобы значок был отображен на экране, значение свойства `Style` элемента управления должно иметь любое значение, отличное от `msoButtonCaption`.

Приведенный ниже оператор устанавливает свойство `FaceId` первой кнопки на панели инструментов `МояПанель` в значение 45, что соответствует коду изображения почтового ящика.

```
CommandBars("МояПанель").Controls(1).FaceId = 45
```

Для определения кода конкретного изображения используется метод проб и ошибок, однако вы можете воспользоваться бесплатной утилитой, которая разработана специально для отображения значения свойства `FaceId`. Эта надстройка упрощает определение свойства `FaceId` конкретного изображения. В листинге 22.3 показана процедура создания панели инструментов с 200 кнопками. Кроме того, панель инструментов содержит раскрывающийся список, с помощью которого можно отобразить на панели инструментов следующие 200 кнопок.

Детально об элементе управления раскрывающегося списка рассказано далее в этой главе.

Листинг 22.3. Создание панели инструментов со всеми кнопками

```
Dim ButtonGroup As Long

Sub ShowFaceIDs()
    Dim NewToolbar As CommandBar
    Dim NewButton As CommandBarButton
    Dim SetNum As CommandBarComboBox
    Dim i As Integer

'   Удаление существующих панелей
    On Error Resume Next
    Application.CommandBars("FaceIds").Delete
    On Error GoTo 0
    ButtonGroup = 1

'   Добавление пустой панели
    Set NewToolbar = Application.CommandBars.Add _
        (Name:="FaceIds", temporary:=True)
    NewToolbar.Visible = True

'   Добавление 200 кнопок
    For i = 1 To 200
        Set NewButton = NewToolbar.Controls.Add _
            (Type:=msoControlButton, ID:=2950)
    Next i

'   Добавление раскрывающегося списка
    Set SetNum = NewToolbar.Controls.Add(Type:=msoControlDropdown)
    For i = 1 To 51
        SetNum.AddItem "Set " & i
    Next i
    With SetNum
        .ListIndex = ButtonGroup
        .Caption = "Button Set Number"
        .OnAction = "NewButtons"
    End With
    NewToolbar.Width = 400

'   Настройка кнопок
    Call NewButtons
End Sub

Sub NewButtons()
'   Изменение кнопок соответственно выбранному в списке значению
    Dim i As Long
    ButtonGroup = CommandBars("FaceIds").Controls("Button Set _
         Number").ListIndex
    For i = 1 To 200
        With CommandBars("FaceIds").Controls(i)
            .FaceId = (ButtonGroup - 1) * 200 + i
```

```
            .Caption = "FaceID = " & (ButtonGroup - 1) * 200 + i
            .OnAction = "EmptySub"
        End With
    Next i
End Sub

Sub EmptySub()
End Sub
```

На рис. 22.11 показана панель инструментов, созданная процедурой `ShowFaceIDs`.

Процедура `ShowFaceIDs` создает панель инструментов и добавляет на нее кнопки. Она вызывает процедуру `NewButtons`, устанавливает свойство `FaceId` и добавляет подпись. Подпись отображается при наведении на кнопку указателя мыши. При выборе другого элемента в раскрывающемся списке выполняется процедура `NewButtons`. Набор отображаемых кнопок определяется переменной `ButtonGroup` уровня модуля.

НА КОМПАКТ-ДИСКЕ

Эта рабочая книга доступна на прилагаемом к книге компакт-диске.

ИСПОЛЬЗОВАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ИЗОБРАЖЕНИЙ

В Excel вы можете использовать огромное количество изображений для кнопок панели инструментов, однако в этой программе существует также возможность нарисовать собственный значок. На рис. 22.12 показан редактор значков для кнопок. Для его запуска выберите команду Вид⇨Панели инструментов⇨Настройка. После отображения на экране диалогового окна щелкните правой кнопкой мыши на кнопке панели инструментов и выберите команду Изменить значок на кнопке.

НОВИНКА

В Excel 2003 в значках используется больше цветов, чем в предыдущих версиях программы.



Рис. 22.11. Эта панель инструментов демонстрирует встроенные кнопки, сгруппированные по 200 штук

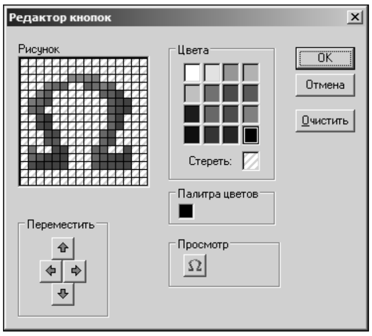


Рис. 22.12. Редактор значков кнопок расширяет ваши изобразительные возможности

ДИНАМИЧЕСКОЕ ИЗМЕНЕНИЕ ПОДПИСИ ЭЛЕМЕНТА УПРАВЛЕНИЯ

Процедура листинга 22.4 создает панель инструментов, которая содержит только одну кнопку. Подпись этой кнопки отображает тип форматирования числа для активной ячейки (рис. 22.13). Процедура использует события объекта `Worksheet` для определения момента изменения выделения. Когда возникает событие `SelectionChange`, процедура выполняется, а подпись кнопки на панели инструментов изменяется.

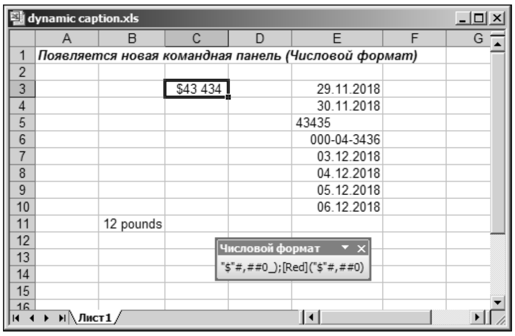


Рис. 22.13. Эта панель инструментов содержит кнопку с форматом числа текущей ячейки

Листинг 22.4. Отображение текущего формата числа активной ячейки

```
Sub MakeNumberFormatDisplay()
    Dim TBar As CommandBar
    Dim NewBtn As CommandBarButton

'   Удаление существующей панели инструментов
    On Error Resume Next
    CommandBars("Числовой формат").Delete
    On Error GoTo 0

'   Создание новой панели инструментов
    Set TBar = CommandBars.Add
    With TBar
        .Name = "Числовой формат"
        .Visible = True
    End With

'   Добавление элемента управления (кнопки)
    Set NewBtn = CommandBars("Числовой формат").Controls.Add _
      (Type:=msoControlButton)
    With NewBtn
        .Caption = ""
        .OnAction = "ChangeNumFormat"
        .TooltipText = "Щелкните для изменения числового формата"
        .Style = msoButtonCaption
        '.Width = 100
    End With
    Call UpdateToolbar
End Sub
```

ПЕРЕКРЕСТНАЯ ССЫЛКА

Дополнительная информация о событиях приведена в главе 19.

Процедура UpdateToolbar, которая отображена ниже, просто копирует свойство NumberFormat объекта ActiveCell в свойство Caption элемента управления (в данном случае — кнопки).

```
Sub UpdateToolbar()
    On Error Resume Next
    CommandBars("Числовой формат"). _
      Controls(1).Caption = ActiveCell.NumberFormat
    If Err <> 0 Then CommandBars("Числовой формат"). _
      Controls(1).Caption = ""
End Sub
```

Свойство OnAction элемента управления (кнопки) указывает на процедуру ChangeNumFormat, которая представлена ниже. Эта процедура отображает вкладку Число диалогового окна Excel Формат ячеек (рис. 22.14).

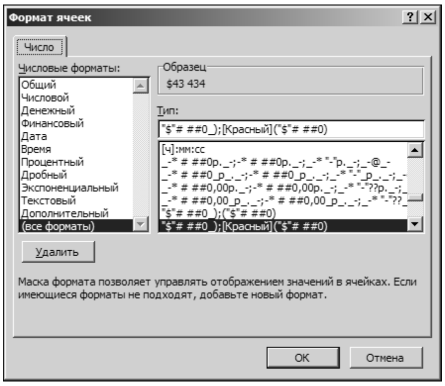


Рис. 22.14. Щелчок на кнопке позволяет пользователю выбрать новый формат отображения чисел

```
Sub ChangeNumFormat()
    Application.Dialogs(xlDialogFormatNumber).Show
    Call UpdateToolbar
End Sub
```

Техника, описанная в этом разделе, работает достаточно неплохо, но имеет один недостаток: если пользователь изменит формат чисел с помощью кнопки на панели инструментов Formatting (Форматирование), то значение подписи кнопки на панели Числовой формат не изменится, так как подобное изменение формата чисел в активной ячейке не приводит к возникновению события, которое можно перехватить.

НАЗНАЧЕНИЕ СОБСТВЕННОГО МАКРОСА ВСТРОЕННОЙ КНОПКЕ

Каждая кнопка на встроенных панелях Excel запускает определенную процедуру. Существует возможность назначить встроенной кнопке собственный макрос. Для этого необходимо воспользоваться свойством OnAction. Следующий оператор назначает макрос кнопке Sort Ascending (Сортировка по возрастанию) на встроенной панели инструментов.

```
CommandBars("Standard").Controls("Sort Ascending") _
  .OnAction = "ShowMsg"
```

После выполнения оператора щелчок на кнопке Sort Ascending (Сортировка по возрастанию) больше не будет приводить к привычному действию. Вместо этого вызывается процедура VBA ShowMsg.

Чтобы вернуть кнопке первоначальную функциональность, назначьте свойству OnAction значение в виде пустой строки.

```
CommandBars("Standard").Controls("Sort Ascending") _
  .OnAction = ""
```

"ВЫЗОВ" КНОПКИ НА КОМАНДНОЙ ПАНЕЛИ

Элементы управления на командной панели имеют метод Execute. После вызова данный метод запускает внутреннюю процедуру, которая назначена встроенному элементу управления. Например, выполнение следующего оператора эквивалентно щелчку на кнопке Sort Ascending (Сортировка по возрастанию) на панели инструментов Standard (Стандартная).

```
CommandBars("Standard").Controls("Sort Ascending").Execute
```

Использование метода Execute по отношению к определенной пользователем кнопке на командной панели приводит к запуску макроса, назначенного с помощью свойства OnAction.

ИСПОЛЬЗОВАНИЕ ДРУГИХ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ КОМАНДНЫХ ПАНЕЛЕЙ

Стандартные кнопки панелей инструментов являются одним из пяти типов элементов управления, которые можно добавлять на панели инструментов. Тип элемента управления определяется значением его свойства Type.

Ниже приведены константы, соответствующие элементам управления. Их *можно* использовать, добавив на командную панель:

- msoControlButton — стандартная кнопка;
- msoControlEdit — текстовое поле;
- msoControlComboBox — комбинированный список;
- msoControlDropdown — раскрывающийся список;
- msoControlButtonPopup — кнопка, которая при щелчке отображает другие элементы управления. С помощью этого элемента управления можно создавать меню, состоящие из нескольких опций.

Свойство Type объекта Control предназначено только для чтения и устанавливается в момент создания элемента управления. Другими словами, изменить тип элемента управления после его создания невозможно.

Процедура MonthList листинга 22.5 создает новую панель инструментов, добавляет на нее элемент управления в виде раскрывающегося списка и заполняет этот элемент управления названиями каждого месяца. Кроме того, определяется свойство OnAction, которое приводит к вызову (в момент щелчка на кнопке) процедуры, вставляющей название месяца. Панель инструментов, которая была получена в результате ее выполнения, показана на рис. 22.15.

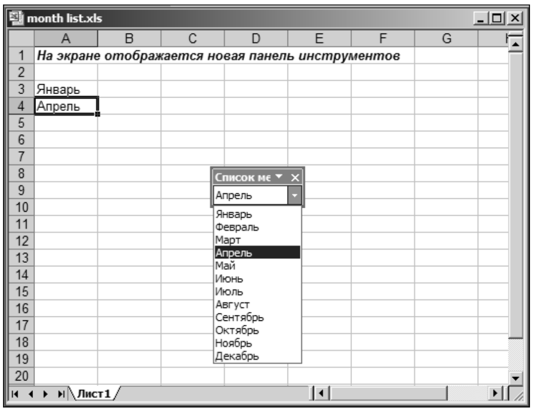


Рис. 22.15. Эта панель инструментов содержит элемент управления (раскрывающийся список), к которому присоединен макрос

Листинг 22.5. Добавление раскрывающегося списка на командную панель

```
Sub MonthList()
    Dim TBar As CommandBar
    Dim NewDD As CommandBarControl

'   Удаление существующей панели инструментов
    On Error Resume Next
    CommandBars("Список месяцев").Delete
    On Error GoTo 0

'   Создание новой панели инструментов
    Set TBar = CommandBars.Add
    With TBar
        .Name = "Список месяцев"
        .Visible = True
    End With

'   Добавление элемента управления DropDown
    Set NewDD = CommandBars("Список месяцев").Controls.Add _
      (Type:=msoControlDropdown)
    With NewDD
        .Caption = "DateDD"
        .OnAction = "PasteMonth"
        .Style = msoButtonAutomatic

'       Заполнение элемента управления названиями месяцев
        For i = 1 To 12
            .AddItem Format(DateSerial(1, i, 1), "mmmm")
        Next i
        .ListIndex = 1
    End With
End Sub
```

Ниже приведена процедура PasteMonth.

```
Sub PasteMonth()
'   Вставляет название выделенного месяца в активную ячейку
    On Error Resume Next
    With CommandBars("Список месяцев").Controls("DateDD")
        ActiveCell.Value = .List(.ListIndex)
    End With
End Sub
```

Данная рабочая книга имеет дополнительную особенность: в ней используется процедура обработки события, которая называется Worksheet_SelectionChange. Эта процедура, которая приводится ниже, выполняется каждый раз, когда пользователь изменяет текущее выделение на рабочем листе. Процедура определяет, не содержит ли активная ячейка название месяца. Если это так, то устанавливается свойство ListIndex раскрывающегося списка на панели инструментов.

```
Private Sub Worksheet_SelectionChange(ByVal Target _
 As Excel.Range)
    For i = 1 To 12
        Set ActCell = Target.Range("A1")
        If ActCell.Value = Format(DateSerial(1, i, 1), _
         "mmmm") Then
            CommandBars("Список месяцев").Controls("DateDD") _
             .ListIndex = i
            Exit Sub
        End If
    Next i
End Sub
```

Глава 23

Создание пользовательских меню

В ЭТОЙ ГЛАВЕ...

Практически каждая программа Windows имеет систему меню, которая служит основным компонентом пользовательского интерфейса программы. Стандарт Windows требует расположения строки меню непосредственно под заголовком окна приложения. Кроме того, некоторые программы предоставляют и другой тип меню — контекстное меню. Обычно щелчок правой кнопкой мыши приводит к отображению контекстного меню, которое содержит часто используемые команды. В этой главе вы познакомитесь со следующими вопросами.

- Обзор меню Excel.
- Типы изменений, которые можно проводить с меню.
- Управление меню с помощью VBA.
- Методы программирования меню, в которых задействованы события.
- Полезная (и очень простая) методика создания пользовательских меню.
- Процедура замены стандартных меню собственными.
- Внесение модификаций в контекстные меню.

Excel использует оба типа меню, что предоставляет разработчикам практически полный контроль над системой меню приложения, включая контекстные меню. Данная глава содержит информацию, необходимую для управления меню в Excel.

Несколько слов о строке меню Excel

Если вы ознакомились с главой 22, то должны знать, что строка меню (как и панель инструментов) является объектом `CommandBar`, а методы, рассмотренные в главе 22, в равной степени относятся и к строке меню.

Чем отличается строка меню от панели инструментов? Строка меню отображается в верхней части диалогового окна приложения Excel, непосредственно под строкой заголовка. При щелчке на строке меню элемент управления верхнего уровня отображает раскрывающийся список элементов второго уровня. Строка меню может содержать три кнопки управления окном (Свернуть окно, Восстановить окно и Закрыть окно), которые отображаются только в том случае, когда окно рабочей книги развернуто.

Панели инструментов обычно состоят из графических значков, на них вы не найдете командных кнопок. Однако перечисленные правила не являются обязательными. Если необходимо, можно добавить традиционные кнопки управления с панелей инструментов в обычное меню или перенести опции меню на обычную панель инструментов. Можно даже переместить строку меню со стандартного места и сделать ее плавающей.

Начиная с Excel 2003 строка меню содержит поле введения вопроса, с помощью которого быстро запускается справочная система программы. Если вам не нравится его присутствие в строке меню, то скройте с помощью следующего VBA-выражения.

```
Application.CommandBars.DisableAskQuestionDropDown = False
```

Операции с меню Excel

Обычным пользователям Excel вполне достаточно стандартных меню. Если вы читаете эту книгу, то значит, вы особенный пользователь. Иногда возникает необходимость внести изменения в существующие меню, чтобы облегчить выполнение собственной работы. Вы также можете редактировать меню, чтобы облегчить работу тех, кто будет использовать разрабатываемое приложение.

В число изменений, которые вносятся в меню Excel, входят удаление, добавление и изменение элементов. Кроме того, можно временно заменить стандартную строку меню Excel на пользовательскую строку меню. Изменения в меню Excel выполняются двумя способами: вручную и с помощью VBA.

При закрытии Excel сохраняются все изменения, которые были внесены в систему меню. Эти изменения отображаются при следующем запуске Excel. Информация об изменениях, внесенных в меню, хранится в файле XLB.

Дополнительная информация о файле XLB приведена в главе 22.

В большинстве случаев не возникает необходимости в сохранении изменений, внесенных в меню. Как правило, требуется создать код VBA, который будет на время вносить изменения в меню, пока открыта определенная рабочая книга. После этого (при закрытии книги) система меню будет восстановлена. Таким образом, вам потребуется не только код VBA, который будет модифицировать меню при открытии рабочей книги, но также код VBA, который восстанавливает систему меню при закрытии этой рабочей книги.

Терминология

Советуем вам ознакомиться с терминологией, которая используется при управлении меню.

- *Командная панель.* Объект, который может функционировать в качестве строки меню, контекстного меню или панели инструментов. Представляется объектом `CommandBar` из библиотеки объектов Microsoft Office.
- *Строка меню.* Строка опций, которая отображается непосредственно под строкой заголовка приложения. Excel имеет две строки меню: одна отображается, когда активен рабочий лист, а вторая — когда активен лист диаграммы или встроенная диаграмма.
- *Меню.* Один элемент верхнего уровня в строке меню. Например, обе строки меню Excel имеют меню Файл.
- *Опция (команда) меню.* Элемент, который отображается в раскрывающемся списке, когда выбирается определенное меню. Например, первая опция меню Файл называется Создать. Опции меню также отображаются в подменю и контекстных меню.
- *Разделительная полоса.* Горизонтальная линия, которая представлена между двумя пунктами меню. Разделительная полоса используется для группирования близких по смыслу опций меню.

- *Подменю.* Меню второго уровня, которое находится в определенном меню. Например, меню Правка имеет подменю Очистить.
- *Опция (команда) подменю.* Опция меню, которая отображается в списке после выбора подменю. Например, подменю Правка⇨Очистить содержит следующие опции подменю: Все, Форматы, Содержимое и Примечания.
- *Контекстное меню.* Всплывающий список опций меню, который отображается при щелчке правой кнопкой мыши на выделении или объекте. Вид и содержимое контекстного меню зависит от выделенного объекта.
- *Активная опция.* Опция меню, которую можно использовать. Если опция меню неактивна, то ее название имеет серый цвет. Такие опции меню применить нельзя.
- *Установленная.* Состояние опции меню, которое обозначает включение или выключение параметра. Установка опции меню отображается специальным флажком, который также может быть сброшен.
- *Значок.* Небольшое графическое изображение, которое располагается возле некоторых опций меню. В терминах VBA с каждым значком связан код, определяемый оператором `FaceID`.
- *Комбинация клавиш для опции меню.* Комбинация клавиш, которая является альтернативным методом выбора опции меню. Комбинация клавиш для опции показана справа от самого элемента меню. Например, комбинация <Ctrl+S> используется для сохранения диалогового окна, которое вызывается по команде Файл⇨Сохранить.

Удаление элементов меню

Можно удалить любой фрагмент системы меню Excel: меню, опции меню и даже целые строки меню. Например, если требуется запретить конечным пользователям получать доступ к параметрам отображения, то следует удалить меню Вид со строки меню листа. Кроме того, можно удалить одну или несколько опций меню. Если удалить опцию Создать из меню Файл, то пользователи не смогут использовать меню для создания новых рабочих книг. Наконец, можно удалить строку меню Excel и заменить ее на собственную строку меню. Таким образом вы позволите контролировать приложение только с помощью созданного макроса.

ВНИМАНИЕ

Важно помнить, что простое удаление строки меню, меню или опций меню никак не повлияет на альтернативные методы выполнения определенных действий. Особенно, если для них представлены комбинации клавиш, кнопки на панелях инструментов, а также команды контекстного меню, которые выполняют те же действия, что и удаленная опция меню. Например, если удалить опцию Создать в меню Файл, то всегда можно использовать кнопку Создать на панели инструментов, комбинацию клавиш <Ctrl+N>, область задач (в Excel 2002–2003) или контекстное меню рабочего стола для создания новой рабочей книги.

Переход от Excel 5/95

Если меню настраивались в Excel 5 или Excel 95, то можете забыть обо всех выученных методах создания таких меню. Начиная с Excel 97, методы внесения изменений в систему меню были существенно изменены.

- *Строка меню фактически является замаскированной панелью инструментов.* Если это утверждение вызывает недоверие, то можно захватить вертикальные полосы в левой части строки меню и перетащить ее в другое место. Если переместить строку меню

достаточно далеко, то в итоге получится плавающая панель инструментов. Официальным обобщающим термином (VBA) для строки меню и панелей инструментов является *командная панель (command bar)*.

- *Исчез редактор меню Excel 5/95*. Для того чтобы отредактировать меню вручную, необходимо воспользоваться командой Вид⇨Панели инструментов⇨Настройка. Но помните, что рабочие книги Excel 5/95, в которых находятся меню, созданные с помощью старых методов редактирования, продолжают работать в Excel 97 и последующих версиях. Внести изменения в подобные меню можно только средствами Excel 5/95.
- *Не существует простого способа назначения макроса VBA новой опции меню Сервис.* В Excel 5/95 это сделать очень просто. Далее в этой главе показан код VBA, который можно использовать для добавления новой опции меню Сервис.
- *Excel 2000 и более поздние версии по умолчанию отображают те опции меню, которые использовались наиболее часто.* На наш взгляд, это одна из наиболее неудачных идей Microsoft. Сложно представить, зачем может понадобится изменять порядок представления опций меню. К счастью, данная возможность отключается с помощью опции вкладки Параметры диалогового окна Настройка.

Добавление элементов меню

Вы вправе добавлять собственные меню во встроенные строки меню. Кроме того, можно добавлять собственные опции меню во встроенные меню и даже создавать новые строки меню, если в этом есть необходимость. Например, при разработке приложения, которое не нуждается во встроенных меню Excel, простым решением будет использование новой строки меню, которая состоит из пользовательских меню и опций меню, запускающих созданные разработчиком макросы. Допускается скрывать стандартную строку меню Excel и заменять ее на созданную разработчиком строку меню.

Изменение опций меню

Если стандартные подписи опций меню Excel вам надоели, то замените их на другие. Например, можно изменить название меню Сервис на Инструменты или Утилиты. Кроме того, встроенной опции меню можно назначить собственный макрос. Существует ряд других возможностей по изменению меню, в число которых входит переорганизация меню в строке меню (например, меню Справка (?) можно отображать в строке первым, а не последним).

Ссылки на коллекцию CommandBars

Коллекция `CommandBars` содержится в объекте `Application`. При ссылке на эту коллекцию в модуле кода VBA общего назначения можно опустить ключевое слово `Application` (подразумевается по умолчанию). Например, следующий оператор (расположенный в модуле кода VBA общего назначения) отображает имя первого элемента коллекции `CommandBars`.

```
MsgBox CommandBars(1).Name
```

При ссылке на коллекцию `CommandBars` из модуля кода объекта `ЭтаКнига` перед ней необходимо установить ссылку на объект `Application`, как показано ниже.

```
MsgBox Application.CommandBars(1).Name
```



При изменении названий встроенных меню Excel следует быть предельно осторожным. Некоторые разработчики Excel предполагают, что меню имеют стандартные названия, поэтому используют эти названия при создании новых опций меню. Код, написанный такими разработчиками, не будет выполняться, если названия меню отличаются от стандартных. Как показано ниже, использование метода `FindControl` помогает избежать подобных проблем.

Далее в этой главе речь пойдет о коде VBA, который предназначен для внесения изменений в существующие меню.

Глава 22 предоставляет полную информацию о диалоговом окне Настройка.

Примеры кода VBA

В настоящем разделе предоставляются практические примеры кода VBA, который предназначен для управления меню Excel.

Вывод информации о меню

Процедура `ListMenuInfo`, которая приведена ниже, демонстрирует описанные принципы. Она отображает название каждого элемента меню (меню, опции меню, а также опции подменю), находящегося в строке меню листа.

```
Sub ListMenuInfo()
    Dim row As Integer
    Dim Menu As CommandBarControl
    Dim MenuItem As CommandBarControl
    Dim SubMenuItem As CommandBarControl
    row = 1
    On Error Resume Next
    For Each Menu In CommandBars(1).Controls
        For Each MenuItem In Menu.Controls
            For Each SubMenuItem In MenuItem.Controls
                Cells(row, 1) = Menu.Caption
                Cells(row, 2) = MenuItem.Caption
                Cells(row, 3) = SubMenuItem.Caption
                row = row + 1
            Next SubMenuItem
        Next MenuItem
    Next Menu
End Sub
```

Оператор `On Error Resume Next` используется с целью не допустить выведение сообщения об ошибке, которое возникает, когда процедура пытается получить доступ к несуществующей опции подменю.

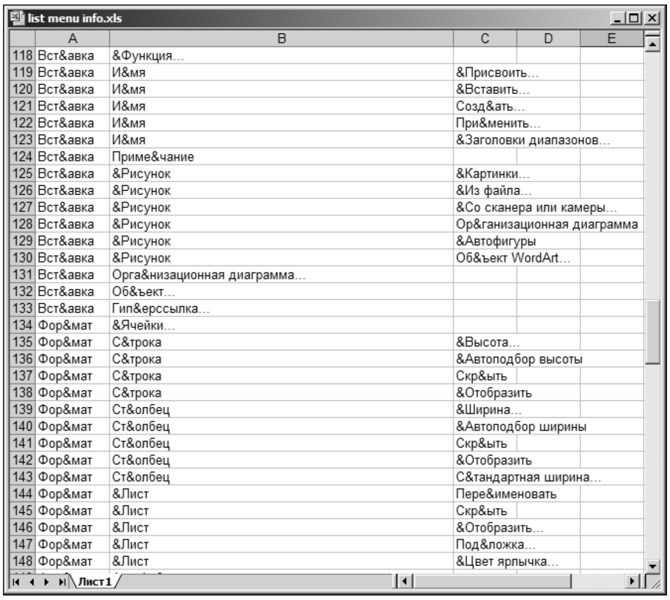
На рис. 23.1 показан результат выполнения процедуры `ListMenuInfo`.

Рабочая книга, которая содержит эту процедуру, доступна на прилагаемом к книге компакт-диске.

Добавление нового меню в строку меню

В этом разделе описан метод использования кода VBA для добавления нового меню в строку меню листа. Строка меню листа является первым элементом коллекции `CommandBars`, поэтому на нее можно ссылаться одним из двух способов.

- По имени: `CommandBars("Worksheet Menu Bar")`.
- По индексу: `CommandBars(1)`.



list menu info.xls

	A	B	C	D	E
118	Вст&авка	&Функция...			
119	Вст&авка	И&мя	&Присвоить...		
120	Вст&авка	И&мя	&Вставить...		
121	Вст&авка	И&мя	Созд&ать...		
122	Вст&авка	И&мя	При&менить...		
123	Вст&авка	И&мя	&Заголовки диапазонов...		
124	Вст&авка	Приме&чание			
125	Вст&авка	&Рисунок	&Картинки...		
126	Вст&авка	&Рисунок	&Из файла...		
127	Вст&авка	&Рисунок	&Со сканера или камеры...		
128	Вст&авка	&Рисунок	Ор&ганизационная диаграмма		
129	Вст&авка	&Рисунок	&Автофигуры		
130	Вст&авка	&Рисунок	Об&ъект WordArt...		
131	Вст&авка	Орга&низационная диаграмма...			
132	Вст&авка	Об&ъект...			
133	Вст&авка	Гип&ерссылка...			
134	Фор&мат	&Ячейки...			
135	Фор&мат	С&трока	&Высота...		
136	Фор&мат	С&трока	&Автоподбор высоты		
137	Фор&мат	С&трока	Скр&ыть		
138	Фор&мат	С&трока	&Отобразить		
139	Фор&мат	Ст&олбец	&Ширина...		
140	Фор&мат	Ст&олбец	&Автоподбор ширины		
141	Фор&мат	Ст&олбец	Скр&ыть		
142	Фор&мат	Ст&олбец	&Отобразить		
143	Фор&мат	Ст&олбец	С&тандартная ширина...		
144	Фор&мат	&Лист	Пере&именовать		
145	Фор&мат	&Лист	Скр&ыть		
146	Фор&мат	&Лист	&Отобразить...		
147	Фор&мат	&Лист	Под&ложка...		
148	Фор&мат	&Лист	&Цвет ярлычка...		

Лист1

Рис. 23.1. Фрагмент, полученный в результате выполнения процедуры `ListMenuItem`

В терминологии VBA метод `Add` используется для добавления нового элемента управления в коллекцию `Controls`. Новый элемент управления является "раскрывающимся"; он имеет тип `msoControlPopup`. Можно указать расположение нового элемента управления. Если этого не сделать, то новый элемент управления будет располагаться в конце строки меню.

Процедура добавления нового меню состоит из двух этапов.

1. Необходимо воспользоваться методом `Add` для добавления переменной, которая представляет новый элемент управления. Аргументы метода `Add` позволяют указать тип элемента управления, идентификатор элемента управления (имеет смысл только в случае добавления встроенного меню), расположение, а также время существования элемента управления (должен ли элемент управления сохраняться в момент закрытия Excel).
2. Необходимо определить свойства добавленного элемента управления. Например, почти всегда требуется указать значения свойств `Caption` и `OnAction`.

Соглашения по созданию меню

Легко заметить, что меню в программах для Windows обычно соответствуют определенным соглашениям. Источники происхождения этих соглашений неизвестны, но им необходимо следовать, если требуется произвести впечатление опытного разработчика, который точно знает, что делает. При внесении изменений в меню необходимо учитывать следующее.

- Традиция требует, чтобы меню Файл всегда располагалось первым, а меню Справка — последним в строке меню.
- Текст меню должен вводиться в правильном регистре. Первая буква каждого названия должна быть прописной.

- Меню верхнего уровня не должны связываться с определенными действиями. Другими словами, каждое меню имеет как минимум одну опцию.
- Подписи опций меню обычно ограничены тремя или меньшим количеством слов.
- Каждая опция меню должна иметь "горячую" клавишу (подчеркнутый символ), которая является уникальной в пределах этого меню.
- Опция меню, которая отображает диалоговое окно, должна завершаться троеточием (...).
- Список опций меню должен оставаться относительно коротким. Иногда подменю являются хорошей альтернативой использованию длинных списков. Если вы все же используете длинный список опций меню, то необходимо вставить разделительные полосы, которые логически группируют сходные опции меню.
- Если это возможно, отключите опции меню, которые не соответствуют текущему контексту. В VBA, чтобы отключить опцию меню (сделать ее неактивной), необходимо изменить значение свойства `Enabled` на `False`.
- Некоторые опции меню служат в качестве переключателей. Когда параметр активен, опция меню отображается с установленным флажком.

ДОБАВЛЕНИЕ МЕНЮ: ПЕРВАЯ ПОПЫТКА

В этом примере главной целью является добавление нового меню Бюджет в строку меню листа. Данное меню должно располагаться слева от меню Справка.

```
Sub AddNewMenu()
    Dim HelpIndex As Integer
    Dim NewMenu As CommandBarPopup

'   Получение индексного номера меню ?
    HelpIndex = CommandBars(1).Controls("Help").Index

'   Создание меню
    Set NewMenu = CommandBars(1).Controls.Add _
      (Type:=msoControlPopup, _
       Before:=HelpIndex, _
       Temporary:=True)

'   Добавление названия
    NewMenu.Caption = "&Бюджет"
End Sub
```

Предыдущий код не является идеальным способом добавления меню. Этот код может вставить, а может и не вставить меню в правильную позицию, что объясняется такими причинами.

- Предполагается, что в строке меню присутствует меню ?. В то же время пользователь мог удалить его.
- Предполагается, что меню ? имеет название *Help*. Неанглийские версии Excel могут использовать другое название для данного меню (в русской версии это *Справка*).

ДОБАВЛЕНИЕ МЕНЮ: ВТОРАЯ ПОПЫТКА

Листинг 23.1 содержит более правильный вариант кода. В представленном коде используется метод `FindControl`, который будет находить меню Справка. Если меню Справка не найдено, то код добавляет новое меню в конец строки меню листа.

Листинг 23.1. Добавление меню Бюджет в строку меню листа Excel

```
Sub AddNewMenu()
    Dim HelpMenu As CommandBarControl
    Dim NewMenu As CommandBarPopup
    Dim MenuItem As CommandBarControl
    Dim Submenuitem As CommandBarButton

'   Удаление меню, если оно существует
    Call DeleteMenu

'   Поиск меню Справка
    Set HelpMenu = CommandBars(1).FindControl(Id:=30010)

    If HelpMenu Is Nothing Then
'       Добавление меню в конец строки меню
        Set NewMenu = CommandBars(1).Controls.Add _
          (Type:=msoControlPopup, _
           temporary:=True)
    Else
'      Добавление меню перед меню Help
        Set NewMenu = CommandBars(1).Controls.Add _
          (Type:=msoControlPopup, _
           Before:=HelpMenu.Index, _
           temporary:=True)
    End If

'   Добавление подписи
    NewMenu.Caption = "&Бюджет"
End Sub
```



Предыдущая процедура создает полностью бесполезное меню (в этом меню отсутствуют функциональные опции). Обратитесь к разделу “Добавление опции в меню Сервис” далее в этой главе для получения дополнительной информации о добавлении опции меню.

Для того чтобы воспользоваться методом `FindControl`, необходимо знать значения свойства `ID` требуемого элемента управления. Каждый встроенный элемент управления Excel `CommandBar` имеет уникальное значение свойства `ID`. В этом примере значение свойства `ID` для меню Справка определено с помощью следующего оператора.

```
MsgBox CommandBars(1).Controls("Справка").ID
```

Окно сообщения отобразило значение 30010 свойства `ID`, которое использовано в качестве параметра метода `FindControl`. В табл. 23.1 показаны значения свойства `ID` для элементов верхнего уровня строки меню Excel.

Таблица 23.1. Значение свойства *ID* меню верхнего уровня Excel

Меню	Значение
File (Файл)	30002
Edit (Правка)	30003
View (Вид)	30004
Insert (Вставка)	30005
Format (Формат)	30006
Tools (Сервис)	30007
Data (Данные)	30011
Chart (Диаграмма)	30022
Window (Окно)	30009
Help (Справка)	30010

Удаление меню из строки меню

Чтобы удалить меню, можно воспользоваться методом `Delete`. Следующий пример демонстрирует способ удаления меню **Бюджет** из строки меню листа. Обратите внимание на использование оператора `On Error Resume Next`, который позволяет игнорировать сообщение об ошибке, возникающее при попытке удалить несуществующее меню.

```
Sub DeleteMenu()
    On Error Resume Next
    CommandBars(1).Controls("Бюджет").Delete
End Sub
```

В процессе создания меню название устанавливалось с помощью следующего оператора.

```
NewMenu.Caption = "&Бюджет"
```

При удалении меню использовать символ амперсанда не обязательно.

Добавление опций в меню

В примере раздела "Добавление нового меню в строку меню" показано, как добавить меню в строку меню. В листинге 23.2 приведен расширенный код, который демонстрирует добавление опций в новое меню.

Листинг 23.2. Добавление опций в меню Бюджет

```
Sub CreateMenu()
    Dim HelpMenu As CommandBarControl
    Dim NewMenu As CommandBarPopup
    Dim MenuItem As CommandBarControl
    Dim Submenuitem As CommandBarButton

'   Удаление меню, если оно существует
    Call DeleteMenu

'   Поиск меню Справка
    Set HelpMenu = CommandBars(1).FindControl(Id:=30010)

    If HelpMenu Is Nothing Then
'       Добавление меню в конец строки меню
        Set NewMenu = CommandBars(1).Controls.Add _
          (Type:=msoControlPopup, _
           temporary:=True)
    Else
'      Добавление меню перед меню Справка
        Set NewMenu = CommandBars(1).Controls.Add _
          (Type:=msoControlPopup, _
           Before:=HelpMenu.Index, _
           temporary:=True)
    End If

'   Добавление подписи
    NewMenu.Caption = "&Бюджет"

'   Первый элемент меню
    Set MenuItem = NewMenu.Controls.Add _
      (Type:=msoControlButton)
    With MenuItem
        .Caption = "&Введение данных..."
        .FaceId = 162
        .OnAction = "Macro1"
    End With
```

```
'   Второй элемент меню
    Set MenuItem = NewMenu.Controls.Add _
      (Type:=msoControlButton)
    With MenuItem
        .Caption = "&Генерация отчета..."
        .FaceId = 590
        .OnAction = "Macro2"
    End With

'   Третий элемент меню
    Set MenuItem = NewMenu.Controls.Add _
      (Type:=msoControlPopup)
    With MenuItem
        .Caption = "Просмотр &диаграмм"
        .BeginGroup = True
    End With

'   Четвертый элемент меню
    Set Submenuitem = MenuItem.Controls.Add _
      (Type:=msoControlButton)
    With Submenuitem
        .Caption = "Ежемесячное изменение"
        .FaceId = 420
        .OnAction = "Macro3"
    End With

'   SECOND SUBMENU ITEM
    Set Submenuitem = MenuItem.Controls.Add _
      (Type:=msoControlButton)
    With Submenuitem
        .Caption = "Отчет за &год"
        .FaceId = 422
        .OnAction = "Macro4"
    End With

End Sub
```

Процедура `CreateMenu` создает меню, показанное на рис. 23.2. Это меню имеет три опции, причем последняя из них представляет собой подменю с двумя опциями.

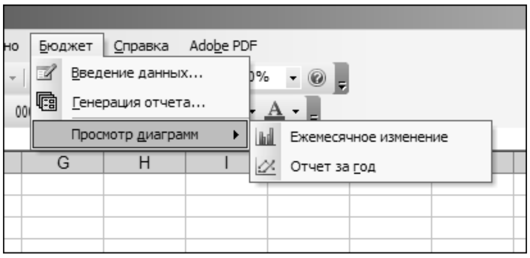


Рис. 23.2. Процедура VBA создает это меню, а также связанные с ним опции меню

НА ЗАМЕТКУ

У вас может возникнуть вопрос: зачем в коде предыдущего примера удалять уже существующее меню — ведь можно завершить выполнение приложения. Воссоздание меню гарантирует использование последней версии меню в строке меню. Кроме того, при разработке приложения таким образом уменьшается количество ручной работы (разработчик освобождается от удаления созданного меню). Процедура создания меню занимает немного времени, поэтому временем, потраченным на воссоздание меню, можно пренебречь.

При рассмотрении процедура CreateMenu помните о следующем.

- Первые две опции меню имеют тип msoControlButton, соответствующий командной кнопке. Третья опция меню имеет тип msoControlPopup, поскольку содержит собственные опции. Таким образом, переменная MenuItem объявлена как имеющая универсальный тип CommandBarControl.
- Свойство BeginGroup третьей опции меню имеет значение True, что обеспечивает отображение разделительной полосы. Разделительная полоса выступает исключительно косметическим элементом управления и служит для группировки функционально близких опций меню.
- Свойство FaceID определяет значок (если он необходим), который будет отображаться возле подписи опции меню. Значение свойства FaceID определяет изображение значка.
- Текст, задающий значение свойства Caption, содержит знак амперсанда (&) для указания символа "горячей" клавиши данной опции меню. Этот символ отображается подчеркнутым и предоставляет доступ к опции меню с помощью клавиатуры.

ДОБАВЛЕНИЕ ОПЦИИ В МЕНЮ СЕРВИС

Пример в листинге 23.2 добавляет несколько опции в пользовательское меню строки меню листа. Часто возникает необходимость в добавлении опций в одно из встроенных меню Excel, например, меню Сервис.

В Excel 5 и Excel 95 назначить макрос новой опции меню Сервис было достаточно просто. По определенным причинам эта возможность отключена, начиная с Excel 97. В данном разделе продемонстрирован метод создания кода VBA, который добавляет опцию во встроенное меню Excel — Сервис.

Листинг 23.3 содержит код добавления опции меню Очистить все кроме формул в меню Сервис. Щелчок на этой опции меню запускает процедуру ClearAllButFormulas.

Листинг 23.3. Добавление опции в меню Сервис

```
Sub AddMenuItem()
    Dim ToolsMenu As CommandBarPopup
    Dim NewMenuItem As CommandBarButton

'   Удаление элемента меню, если он существует
    Call DeleteMenuItem

'   Поиск меню Сервис
    Set ToolsMenu = CommandBars(1).FindControl(Id:=30007)
    If ToolsMenu Is Nothing Then
        MsgBox "Невозможно добавить элемент."
        Exit Sub
    Else
        Set NewMenuItem = ToolsMenu.Controls.Add _
          (Type:=msoControlButton)
        With NewMenuItem
            .Caption = "Очистить в&се кроме формул"
            .FaceId = 348
            .OnAction = "ClearAllButFormulas"
            .BeginGroup = True
        End With
    End If
End Sub
```

На рис. 23.3 показано меню Сервис с новой опцией. Обратите внимание, что код не ссылается на меню Сервис по названию. Вместо этого данное меню определяется с помощью свойства ID (которое имеет значение 30007).

Приведенный выше пример доступен на прилагаемом к книге компакт-диске.

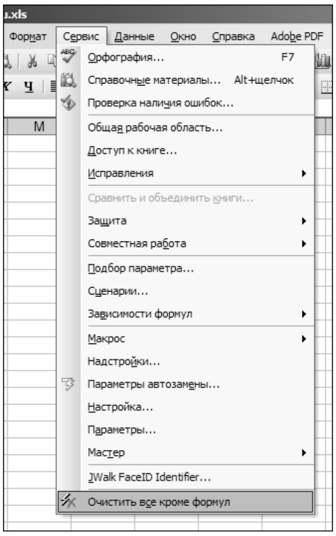


Рис. 23.3. В меню Сервис добавлена новая опция

УДАЛЕНИЕ ОПЦИИ ИЗ МЕНЮ СЕРВИС

Для того чтобы удалить опцию меню, необходимо воспользоваться методом Delete коллекции Controls. В следующем примере удаляется опция Очистить все кроме формул меню Сервис. Обратите внимание, что метод FindControl используется для обработки такой ситуации, когда меню Сервис имеет другое название.

```
Sub DeleteMenuItem()
    On Error Resume Next
    CommandBars(1).FindControl(Id:=30007). _
    Control("Очистить в&се кроме формул").Delete
End Sub
```

Отображение комбинации клавиш вместе с опцией меню

Некоторым встроенным опциям меню Excel назначаются определенные комбинации клавиш. Например, меню Правка содержит несколько опций с комбинациями клавиш.

Для того чтобы отобразить комбинацию клавиш возле опции меню, необходимо воспользоваться свойством ShortcutText. Важно понимать, что установка свойства ShortcutText не приводит к назначению комбинации клавиш. Установка этого свойства просто отобразит назначенную комбинацию клавиш в названии опции меню. Чтобы фактически назначить комбинацию клавиш определенной команде, необходимо сгенерировать дополнительный код VBA.

Код листинга 23.4 создает опцию Очистить все кроме формул меню Сервис. Кроме того, значение свойства ShortcutText этой опции меню устанавливается равным "Ctrl+Shift+C". В коде также используется метод MacroOptions, который позволяет назначить указанную комбинацию клавиш.

Листинг 23.4. Добавление опции меню с комбинацией клавиш

```
Sub AddMenuItem()
    Dim ToolsMenu As CommandBarPopup
    Dim NewMenuItem As CommandBarButton

'   Удаление элемента меню, если он существует
    Call DeleteMenuItem

'   Поиск меню Сервис
    Set ToolsMenu = CommandBars(1).FindControl(Id:=30007)
    If ToolsMenu Is Nothing Then
        MsgBox "Невозможно добавить элемент - используйте Ctrl+Shift+C."
        Exit Sub
    Else
        Set NewMenuItem = ToolsMenu.Controls.Add _
          (Type:=msoControlButton)
        With NewMenuItem
            .Caption = "Очистить в&се, кроме формул"
            .FaceId = 348
            .ShortcutText = "Ctrl+Shift+C"
            .OnAction = "ClearAllButFormulas"
            .BeginGroup = True
        End With
    End If

'   Назначение комбинации клавиш
    Application.MacroOptions _
      Macro:="ClearAllButFormulas", _
      HasShortcutKey:=True, _
      ShortcutKey:="C"

End Sub
```

После выполнения этой процедуры опция меню будет отображаться так, как показано на рис. 23.4.

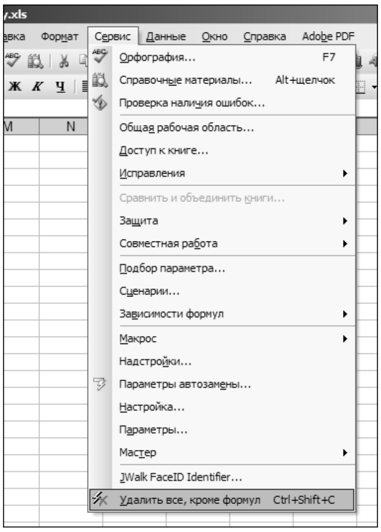


Рис. 23.4. Опция меню Очистить все, кроме формул содержит комбинацию клавиш

Исправление восстановленного меню

Представим себе такой сценарий развития событий. Вы написали код VBA, создающий меню при открытии рабочей книги приложения. Пользователь открывает новую рабочую книгу, которая содержит макрос, восстанавливающий состояние строки меню. Можно также рассмотреть следующий вариант: пользователь работает с диалоговым окном Настройка, выбирает опцию Строка меню листа в списке этого диалогового окна и щелкает на кнопке Сброс. В обоих случаях добавленное меню будет удалено.

Код создания меню обычно запускается по происхождению события `Workbook_Open`, поэтому единственным способом повторного получения модифицированного меню является закрытие и открытие рабочей книги приложения. Для того чтобы предоставить еще один способ получения меню, можно добавить комбинацию клавиш, которая будет повторно запускать процедуру создания меню.

Очевидно, что приложения, восстанавливающие состояние строки меню Excel, достаточно распространены. Пользователи пакета Power Utility Pak часто замечают, что меню PUP 2000 исчезает без всякой причины. Обычно это происходит потому, что другое приложение восстанавливает строку меню листа. В связи с этим в пакет была добавлена комбинация клавиш <Ctrl+Shift+U>, которая должна добавить в строку меню PUP 2000. При выполнении представленного оператора процедура `CreateMenu` связывается с комбинацией клавиш <Ctrl+Shift+U>.

```
Application.MacroOptions Macro:="CreateMenu", _
     HasShortcutKey:=True, ShortcutKey:="U"
```

Работа с событиями

Предположим, вам необходимо создать меню в момент открытия рабочей книги. Кроме того, после закрытия рабочей книги требуется удалить это меню, так как многие модификации, внесенные в него, продолжают существовать и после закрытия Excel. Также иногда возникает необходимость в создании меню, доступного только при активизации определенной рабочей книги или рабочего листа. Такие действия можно запрограммировать, поскольку в них легко использовать события, обрабатываемые в Excel.

Примеры, приведенные в этом разделе, демонстрируют различные способы программирования меню, которые используются совместно с событиями.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Программирование событий подробно рассматривалось в главе 19.

Автоматическое добавление и удаление меню

Если необходимо, чтобы меню добавлялось при открытии рабочей книги, то следует воспользоваться событием `Open` объекта `Workbook`. Следующий код, который находится в модуле кода объекта `ЭтаКнига`, запускает процедуру `CreateMenu`.

```
Private Sub Workbook_Open()
    Call CreateMenu
End Sub
```

Для того чтобы удалить меню при закрытии рабочей книги, воспользуйтесь процедурой, которая приведена ниже. Эта процедура выполняется перед закрытием рабочей книги, она вызывает процедуру `DeleteMenu`.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Call DeleteMenu
End Sub
```

Но в данном случае может возникнуть проблема, если при закрытии рабочая книга не сохраняется. Окно с сообщением `Сохранить изменения...?` отображается уже после того, как возникнет событие `BeforeClose`, а процедура `Workbook_BeforeClose` завершит свою работу. Поэтому, если пользователь щелкнет на кнопке Отмена, рабочая книга останется открытой, а меню уже будет удалено!

Одним из решений этой проблемы является обход сообщения Excel и создание собственного кода, который выдает запрос на сохранение изменений в рабочей книге. Этот код должен располагаться в процедуре `Workbook_BeforeClose`.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    If Not Me.Saved Then
        Msg = "Сохранить изменения в"
        Msg = Msg & Me.Name & "?"
        Ans = MsgBox(Msg, vbQuestion + vbYesNoCancel)
        Select Case Ans
            Case vbYes
                Me.Save
            Case vbNo
                Me.Saved = True
            Case vbCancel
                Cancel = True
                Exit Sub
        End Select
    End If
    Call DeleteMenu
End Sub
```

Представленная процедура определяет, была ли сохранена рабочая книга. Если это так, то никаких проблем не возникает. Выполняется процедура `DeleteMenu`, и рабочая книга закрывается. Но если рабочая книга не была сохранена, то процедура отображает такое окно сообщения, которое дублирует окно сообщения, по умолчанию отображаемое Excel. Если пользователь щелкнет на кнопке Да, то рабочая книга будет сохранена, меню удалено, а рабочая книга закроется. Если пользователь щелкает на кнопке Нет, то свойство `Saved` рабочей книги будет установлено в значение `True` (но фактически изменения в рабочей книге не сохраняются) и меню будет удалено. Если пользователь щелкнет на кнопке Отмена, то событие `BeforeClose` отменяется, а процедура завершается без удаления меню.

Отключение или скрытие меню

Когда меню или опция меню отключены, то соответствующая подпись будет представлена светло-серым цветом. Щелчок на таком элементе меню ни к чему не приводит. Excel отключает опции меню, если они неприменимы в текущем контексте. Например, опция Связи меню Правка отключена, если активная рабочая книга не содержит связей.

Можно создать код VBA, который будет включать или отключать встроенные и пользовательские меню, а также опции меню. Точно так же можно создать код, который скрывает меню или опции меню. Ключевым моментом в создании такого кода является использование правильного события.

Следующая процедура сохраняется в модуле кода объекта `ЭтаКнига`.

```
Private Sub Workbook_Open()
    Call AddMenu
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Call DeleteMenu
End Sub
```

```
Private Sub Workbook_Activate()
    Call UnhideMenu
End Sub

Private Sub Workbook_Deactivate()
    Call HideMenu
End Sub
```

Когда открывается рабочая книга, вызывается процедура AddMenu. Если рабочая книга закрывается, вызывается процедура DeleteMenu. Две дополнительные процедуры обработки событий выполняются, когда рабочая книга активизируется или деактивизируется. Процедура UnhideMenu вызывается, когда рабочая книга активизируется, а процедура HideMenu вызывается в том случае, если рабочая книга деактивизируется.

Процедура HideMenu устанавливает значение свойства Visible опции меню равным False, что приводит к его скрытию. Процедура UnhideMenu выполняет противоположную функцию. Общим результатом является отображение опции меню только в том случае, когда рабочая книга является активной. Процедуры, в которых предполагается, что свойство Caption меню равно Бюджет, приводятся ниже.

```
Sub UnhideMenu()
    CommandBars(1).Controls("Бюджет").Visible = True
End Sub

Sub HideMenu()
    CommandBars(1).Controls("Бюджет").Visible = False
End Sub
```

Для того чтобы отключить меню, а не скрыть его, необходимо вместо свойства Visible изменить значение свойства Enabled.

НА КОМПАКТ-ДИСКЕ

Данный пример находится на прилагаемом к книге компакт-диске.

Работа с установленными опциями меню

Некоторые встроенные опции меню Excel могут отображаться с установленным флажком или без него. Например, опция меню Вид⇨Строка формул имеет установленный флажок, если панель формул отображается на экране. Если панель формул скрыта, то флажок возле этой опции меню отсутствует. При выборе данной опции состояние панели формул изменяется, и флажок выставляется.

Такую функциональность можно добавить и к пользовательским опциям меню. На рис. 23.5 показана опция меню, которая имеет флажок только тогда, когда активный лист содержит линии сетки. Выбор такой опции меню изменяет состояние отображения линий разметки, а также состояние флажка напротив опции меню. Присутствие флажка зависит от значения свойства State элемента управления опции меню.

В рассматриваемом случае основной задачей является синхронизация состояния флажка с активностью листа. Чтобы обеспечить это, потребуется обновлять опцию меню каждый раз, когда активизируется новый лист, новая рабочая книга или новое окно. Необходимые действия выполняются с помощью событий уровня приложения.

ДОБАВЛЕНИЕ ОПЦИИ МЕНЮ

Процедура AddMenuItem, показанная в листинге 23.5, выполняется при каждом открытии рабочей книги. Эта процедура создает новую опцию Линии сетки в меню Вид.

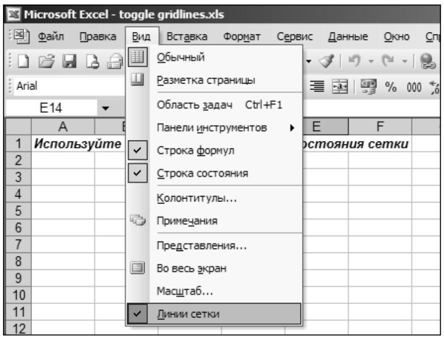


Рис. 23.5. Опция меню Линии сетки имеет установленный флажок, если активный лист содержит линии сетки

Листинг 23.5. Внесение изменений во встроенное меню Excel

```
Dim AppObject As New XLHandler


Sub AddMenuItem()
    Dim ViewMenu As CommandBarPopup
    Dim NewMenuItem As CommandBarButton

'   Удаление опции, если она существует
    Call DeleteMenuItem

'   Поиск меню Вид
    Set ViewMenu = CommandBars(1).FindControl(ID:=30004)
    If ViewMenu Is Nothing Then
        MsgBox "Невозможно добавить элемент меню."
        Exit Sub
    Else
        Set NewMenuItem = ViewMenu.Controls.Add _
          (Type:=msoControlButton)
        With NewMenuItem
            .Caption = "&Линии сетки"
            .OnAction = "ToggleGridlines"
        End With
    End If

'   Обработчик события
    Set AppObject.AppEvents = Application
End Sub
```

Процедура AddMenuItem добавляет новую опцию в строку меню листа, а не в строку меню диаграммы. По этой причине новая опция отображается только тогда, когда активен рабочий лист (что является обязательным требованием).

Обратите внимание: в последнем операторе процедура AddMenuItem настраивает события уровня приложения, которые будут обрабатываться в приложении. Процедуры обработки событий, которые хранятся в модуле класса XLHandler, приведены ниже.

```
Public WithEvents AppEvents As Application

Private Sub AppEvents_SheetActivate(ByVal Sh As Object)
    Call CheckGridlines
End Sub
```

```
Private Sub AppEvents_WorkbookActivate(ByVal Wb As Excel.Workbook)
    Call CheckGridlines
End Sub

Private Sub AppEvents_WindowActivate _
  (ByVal Wb As Workbook, ByVal Wn As Window)
    Call CheckGridlines
End Sub
```

Представленная процедура имеет один недостаток: изменение состояния отображения линий сетки с помощью диалогового окна Параметры не отслеживается.

ОТОБРАЖЕНИЕ ЛИНИЙ СЕТКИ

Главным эффектом является запуск процедуры `CheckGridlines` при активизации пользователем другой рабочей книги или другого рабочего листа. Эта процедура обеспечивает состояние флажка для опции Линии сетки меню Вид в соответствии с параметрами активного листа.

```
Sub CheckGridlines()
    Dim TG As CommandBarButton
    On Error Resume Next
    Set TG = CommandBars(1).FindControl(ID:=30004). _
      Controls("&Линии сетки")
    If ActiveWindow.DisplayGridlines Then
        TG.State = msoButtonDown
    Else
        TG.State = msoButtonUp
    End If
End Sub
```

Данная процедура проверяет характеристики активного окна и устанавливает соответствующее значение свойства `State` опции меню. Если линии сетки отображаются, то к опции Линии сетки добавляется флажок. Если линии сетки не отображаются, то флажок из соответствующей опции меню удаляется.

СИНХРОНИЗАЦИЯ МЕНЮ С АКТИВНЫМ ЛИСТОМ

Если выбрана опция меню, значение свойства `OnAction` этой опции изменяется, что приводит к вызову процедуры `ToggleGridlines`, которая показана ниже.

```
Sub ToggleGridlines()
    If TypeName(ActiveSheet) = "Worksheet" Then
        ActiveWindow.DisplayGridlines = _
           Not ActiveWindow.DisplayGridlines
        Call CheckGridlines
    End If
End Sub
```

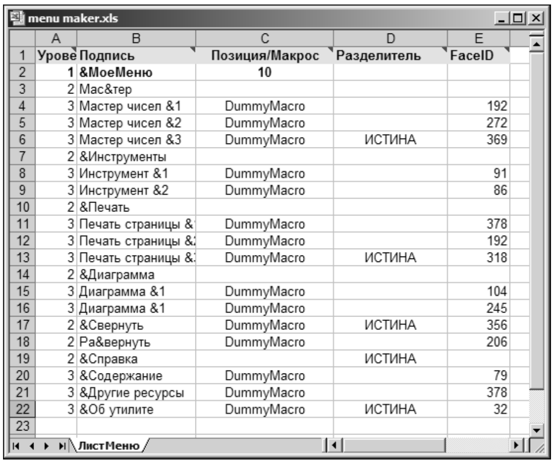
Эта процедура просто переключает состояние линий сетки в активном листе. Конструкция `If-Then` используется для того, чтобы проверить принадлежность активного листа к рабочим листам.

Простой способ создания пользовательских меню

Когда появилась программа Excel 97, многих удивило то количество кода, которое необходимо было писать для создания пользовательских меню. В результате мне пришлось разработать собственную методику, которая упрощает этот процесс. Данная методика требует использования рабочего листа, который показан на рис. 23.6. Этот лист необходим для хранения информации о новом меню. Процедура VBA считывает данные из рабочего листа и создает меню, опции меню и опции подменю.

Рабочий лист состоит из таблицы, которая имеет пять столбцов.

- *Уровень.* Это расположение определенной опции в иерархии системы меню. Допустимыми значениями данного столбца выступают целые числа 1, 2 и 3. Уровень 1 соответствует меню. Уровень 2 описывает опции меню. Уровень 3 предназначен для доступа к опциям подменю. Обычно создается один элемент первого уровня, а для него — несколько элементов второго уровня. Опции второго уровня могут иметь, а могут и не иметь элементы третьего уровня (опции подменю).
- *Подпись.* Этот текст отображается в качестве подписи меню, опции меню или подменю. Чтобы подчеркнуть один из символов в названии, необходимо перед ним ввести символ амперсанда (&).
- *Позиция/Макрос.* Для элементов первого уровня это значение должно быть представлено целым числом, которое обозначает расположение элемента в строке меню. Для элементов второго и третьего уровня это название макроса, который будет выполняться при выборе данной опции меню. Если элемент второго уровня имеет один или несколько элементов третьего уровня, то с элементом второго уровня макрос связать нельзя.
- *Разделитель.* Введите `ИСТИНА`, если перед опцией меню или подменю должна располагаться разделительная полоса.
- *FaceID.* Этот необязательный параметр содержит код, который представляет встроенное графическое изображение, отображаемое возле опции меню.


menu maker.xls

	A	B	C	D	E
1	Урове	Подпись	Позиция/Макрос	Разделитель	FaceID
2	1	&МоеМеню	10		
3	2	Мас&тер			
4	3	Мастер чисел &1	DummyMacro		192
5	3	Мастер чисел &2	DummyMacro		272
6	3	Мастер чисел &3	DummyMacro	ИСТИНА	369
7	2	&Инструменты			
8	3	Инструмент &1	DummyMacro		91
9	3	Инструмент &2	DummyMacro		86
10	2	&Печать			
11	3	Печать страницы &	DummyMacro		378
12	3	Печать страницы &	DummyMacro		192
13	3	Печать страницы &	DummyMacro	ИСТИНА	318
14	2	&Диаграмма			
15	3	Диаграмма &1	DummyMacro		104
16	3	Диаграмма &1	DummyMacro		245
17	2	&Свернуть	DummyMacro	ИСТИНА	356
18	2	Ра&вернуть	DummyMacro		206
19	2	&Справка		ИСТИНА	
20	3	&Содержание	DummyMacro		79
21	3	&Другие ресурсы	DummyMacro		378
22	3	&Об утилите	DummyMacro	ИСТИНА	32
23					

ЛистМеню

Рис. 23.6. Информация этого рабочего листа используется для создания меню

На рис. 23.7 показано меню, которое создано с помощью данных, сохраненных на рабочем листе.

Рабочая книга, демонстрирующая эту методику, доступна на прилагаемом к книге компакт-диске. Эта рабочая книга содержит процедуру VBA, которая считывает данные с рабочего листа и создает меню. Для того чтобы использовать эту методику в собственной рабочей книге или в надстройке, необходимо следовать инструкциям, приведенным ниже.

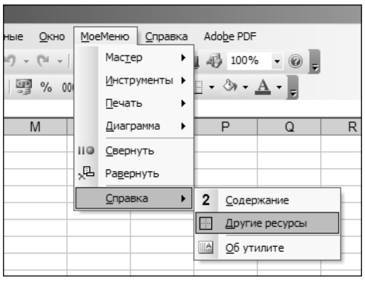


Рис. 23.7. Меню, созданное на основе данных, которые хранились на рабочем листе

1. Откройте пример рабочей книги, который находится на прилагаемом компакт-диске.
2. Скопируйте весь код из модуля `Module1` в модуль собственного проекта.
3. Добавьте процедуры, которые приведены ниже, к модулю кода объекта `ЭтаКнига`.

```
Private Sub Workbook_Open()
    Call CreateMenu
End Sub
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Call DeleteMenu
End Sub
```

4. Вставьте новый рабочий лист и назовите его `ЛистМеню`. Еще более удачным вариантом будет копирование рабочего листа `ЛистМеню` из примера рабочей книги.
5. Внесите изменения в рабочий лист `ЛистМеню`, чтобы он соответствовал пользовательскому меню.

В примере рабочей книги не реализована обработка сообщений об ошибках, поэтому на разработчика возлагается задача проверки правильности создания меню.

Замена строки меню листа

В некоторых случаях может возникнуть необходимость скрыть строку меню листа Excel и заменить ее полностью новой строкой меню.

Процедура `MakeMenuBar` из листинга 23.6 создает новую строку меню, которая называется `Моя строка меню`. Эта строка состоит из двух меню. Первое меню повторяет стандартное меню Файл (оно скопировано со строки меню листа). Второе меню содержит две опции: Восстановить обычное меню и Справка (?).

Листинг 23.6. Замена встроенного меню Excel

```
Sub MakeMenuBar()
    Dim NewMenuBar As CommandBar
    Dim NewMenu As CommandBarControl
    Dim NewItem As CommandBarControl

'   Удаление строки меню, если она существует
    Call DeleteMenuBar

'   Добавление строки меню
    Set NewMenuBar = CommandBars.Add(MenuBar:=True)
    With NewMenuBar
        .Name = "Моя строка меню"
        .Visible = True
    End With

'   Копирование меню File (ID=30002) со строки меню листа
    CommandBars("Worksheet Menu Bar").FindControl(ID:=30002).Copy _
      Bar:=CommandBars("Моя строка меню")

'   Добавление нового меню
    Set NewMenu = NewMenuBar.Controls.Add _
      (Type:=msoControlPopup)
    NewMenu.Caption = "&Команды"

'   Добавление новой опции меню
    Set NewItem = NewMenu.Controls.Add(Type:=msoControlButton)
    With NewItem
        .Caption = "&Восстановить обычную строку меню"
        .OnAction = "DeleteMenuBar"
    End With

'   Добавление новой опции меню
    Set NewItem = NewMenu.Controls.Add(Type:=msoControlButton)
    With NewItem
        .Caption = "&Справка"
        .OnAction = "ShowHelp"
    End With
End Sub
```

На рис. 23.8 показана новая строка меню.

Обратите внимание, что в этой процедуре отсутствует код, который скрывает строку меню листа. Оператор `Set NewMenuBar = CommandBars.Add(MenuBar:=True)` добавляет новую командную панель. Когда свойство `Visible` этой командной панели устанавливается в значение `True`, она заменяет собой строку меню рабочего листа. В определенный момент времени может быть активной только одна строка меню.

Стандартная панель инструментов (как отмечалось в главе 22) имеет свойство `Type`, по умолчанию установленное в значение `msoBarTypeNormal`. Строка меню, которая создавалась в предыдущем примере, имеет свойство `Type`, установленное в значение `msoBarTypeMenuBar`.

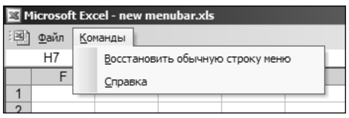


Рис. 23.8. Созданная строка меню заменяет стандартную строку меню листа

Удаление пользовательской строки меню приводит к отображению строки меню листа и делает эту строку активной. Следующая процедура (DeleteMenuBar) возвращает строку меню в исходное состояние.

```
Sub DeleteMenuBar()
    On Error Resume Next
    CommandBars("Моя строка меню").Delete
    On Error GoTo 0
End Sub
```

Следующий код предназначен для нахождения элемента управления Файл командной панели с помощью метода FindControl. Это меню копируется из строки меню листа на новую строку меню с помощью метода Copy.

```
CommandBars("Worksheet Menu Bar").FindControl(ID:=30002).Copy _
         Bar:= CommandBars ("My Menu Bar")
```

Когда выполняется этот оператор, меню Файл (вместе со всеми опциями и подменю) отображается в новой строке меню. Следует отметить, что опции меню Файл и его подменю не являются точными копиями оригиналов в строке меню листа. Например, измените свойство Caption опции Создать в строке MyMenuBar (замените название с *New (Создать)* на *New Workbook (Создать рабочую книгу)*). Это приведет также к изменению опции Создать в строке меню листа. В результате после восстановления строки меню листа в меню Файл будет отображена опция с измененным названием.

Работа с контекстными меню

Контекстное меню является всплывающим меню, которое отображается при щелчке правой кнопкой мыши на практически каждом объекте в Excel. Диалоговое окно Excel Настройка не предоставляет возможностей по редактированию или удалению контекстных меню. Единственным способом внесения модификаций в контекстные меню является использование VBA.

Контекстное меню представлено командной панелью, свойство Type которой установлено в значение msoBarTypePopup. Для того чтобы управлять контекстным меню, необходимо знать значение его свойства Index или Name. Представленную ниже процедуру можно использовать для создания списка всех контекстных меню. Этот список отображает информацию о каждом контекстном меню на рабочем листе. При этом в столбцах представлена информация о значениях свойств Index Name, а также список всех опций этого меню.

```
Sub ListShortCutMenus()
    Dim Row As Long, Col As Integer
    Dim cbar As CommandBar

    Cells.Clear
    Application.ScreenUpdating = False
    Row = 1
    For Each cbar In CommandBars
        If cbar.Type = msoBarTypePopup Then
            Cells(Row, 1) = cbar.Index
            Cells(Row, 2) = cbar.Name
            For Col = 1 To cbar.Controls.Count
                Cells(Row, Col + 2) = _
                 cbar.Controls(Col).Caption
            Next Col
            Row = Row + 1
        End If
    Next cbar
    Cells.EntireColumn.AutoFit
End Sub
```

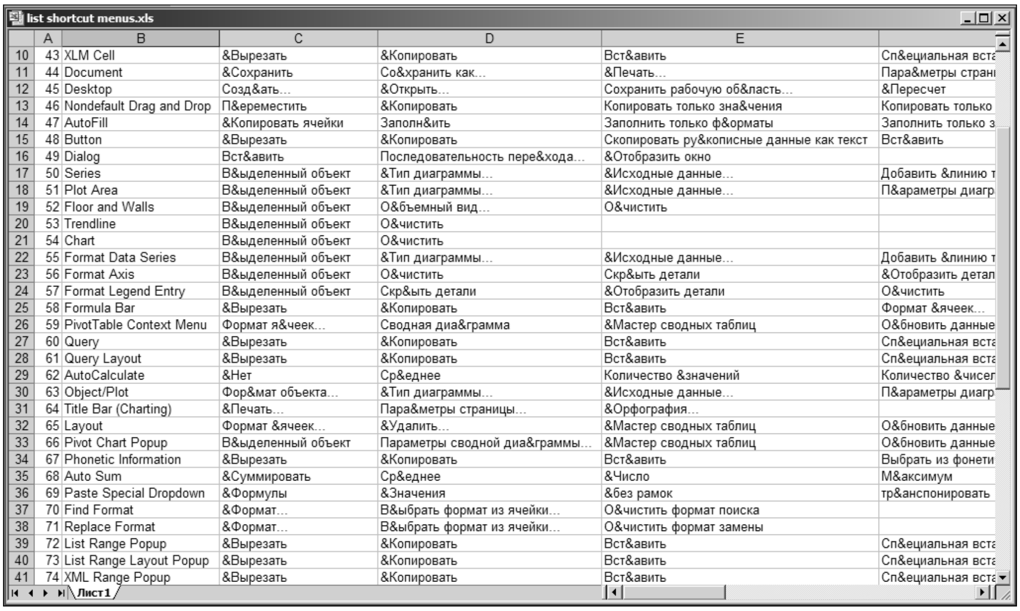
На рис. 23.9 показан фрагмент результата выполнения этой процедуры. Процедура помогает определить имена различных контекстных меню. Например, кто может догадаться, что контекстное меню, которое отображается при щелчке правой кнопкой мыши на ярлыке рабочего листа, называется *Ply*?

Несмотря на то, что на контекстное меню можно ссылаться по значению свойства `Index`, этот метод использовать не рекомендуется. По определенным причинам значения свойства `Index` не одинаковы в разных версиях Excel. Вместо этого для создания ссылки на контекстное меню необходимо использовать значение свойства `Name`.

Добавление опций в контекстное меню

Добавление опции в контекстное меню выполняется подобно вставке опции в обычное меню. Следующий пример демонстрирует добавление опции в контекстное меню Cell (Ячейка), которое отображается, когда пользователь щелкает правой кнопкой мыши на ячейке, на строке или на границе столбца. Эта опция добавляется в конец контекстного меню, сразу за разделительной полосой.

```
Sub AddItemToShortcut()
    Set NewItem = CommandBars("Cell").Controls.Add
    With NewItem
       .Caption = "Перенос слов"
       .OnAction = "ToggleWordWrap"
       .BeginGroup = True
    End With
End Sub
```



list shortcut menus.xls

	A	B	C	D	E	
10	43	XLM Cell	&Вырезать	&Копировать	Вст&авить	Сп&ециальная вста
11	44	Document	&Сохранить	Со&хранить как...	&Печать...	Пара&метры страни
12	45	Desktop	Созд&ать...	&Открыть...	Сохранить рабочую об&ласть...	&Пересчет
13	46	Nondefault Drag and Drop	П&ереместить	&Копировать	Копировать только зна&чения	Копировать только
14	47	AutoFill	&Копировать ячейки	Заполн&ить	Заполнить только ф&орматы	Заполнить только з
15	48	Button	&Вырезать	&Копировать	Скопировать ру&кописные данные как текст	Вст&авить
16	49	Dialog	Вст&авить	Последовательность пере&хода...	&Отобразить окно	
17	50	Series	В&ыделенный объект	&Тип диаграммы...	&Исходные данные...	Добавить &линию т
18	51	Plot Area	В&ыделенный объект	&Тип диаграммы...	&Исходные данные...	П&араметры диагр
19	52	Floor and Walls	В&ыделенный объект	О&бъемный вид...	О&чистить	
20	53	Trendline	В&ыделенный объект	О&чистить		
21	54	Chart	В&ыделенный объект	О&чистить		
22	55	Format Data Series	В&ыделенный объект	&Тип диаграммы...	&Исходные данные...	Добавить &линию т
23	56	Format Axis	В&ыделенный объект	О&чистить	Скр&ыть детали	&Отобразить детал
24	57	Format Legend Entry	В&ыделенный объект	Скр&ыть детали	&Отобразить детали	О&чистить
25	58	Formula Bar	&Вырезать	&Копировать	Вст&авить	Формат &ячеек...
26	59	PivotTable Context Menu	Формат я&чеек...	Сводная диа&грамма	&Мастер сводных таблиц	О&бновить данные
27	60	Query	&Вырезать	&Копировать	Вст&авить	Сп&ециальная вста
28	61	Query Layout	&Вырезать	&Копировать	Вст&авить	Сп&ециальная вста
29	62	AutoCalculate	&Нет	Ср&еднее	Количество &значений	Количество &чисел
30	63	Object/Plot	Фор&мат объекта...	&Тип диаграммы...	&Исходные данные...	П&араметры диагр
31	64	Title Bar (Charting)	&Печать...	Пара&метры страницы...	&Орфография...	
32	65	Layout	Формат &ячеек...	&Удалить...	&Мастер сводных таблиц	О&бновить данные
33	66	Pivot Chart Popup	В&ыделенный объект	Параметры сводной диа&граммы...	&Мастер сводных таблиц	О&бновить данные
34	67	Phonetic Information	&Вырезать	&Копировать	Вст&авить	Выбрать из фонети
35	68	Auto Sum	&Суммировать	Ср&еднее	&Число	М&аксимум
36	69	Paste Special Dropdown	&Формулы	&Значения	&без рамок	тр&анспонировать
37	70	Find Format	&Формат...	В&ыбрать формат из ячейки...	О&чистить формат поиска	
38	71	Replace Format	&Формат...	В&ыбрать формат из ячейки...	О&чистить формат замены	
39	72	List Range Popup	&Вырезать	&Копировать	Вст&авить	Сп&ециальная вста
40	73	List Range Layout Popup	&Вырезать	&Копировать	Вст&авить	Сп&ециальная вста
41	74	XML Range Popup	&Вырезать	&Копировать	Вст&авить	Сп&ециальная вста

Лист1

Рис. 23.9. Список всех контекстных меню, а также список всех опций, которые входят в эти контекстные меню

Выбор новой опции меню приводит к вызову процедуры ToggleWordWrap. На рис. 23.10 показано обновленное контекстное меню.

В предыдущем примере использовалось свойство OnAction с целью назначить макрос опции контекстного меню. Представленный далее пример не требует использования свойства OnAction. Вместо этого в контекстное меню добавляется встроенная команда Скрыть окно. Данное контекстное меню отображается, когда пользователь щелкает правой кнопкой мыши на строке заголовка окна рабочей книги.

```
Sub AddItemToShortcut()
    Set NewItem = CommandBars("Document").Controls.Add(ID:=865)
    NewItem.Caption = "Скрыть окно"
End Sub
```

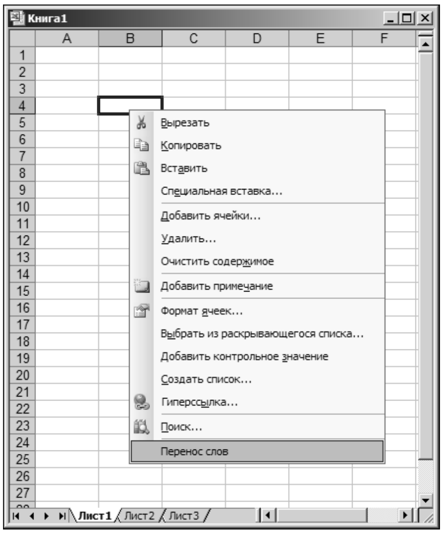


Рис. 23.10. Это контекстное меню содержит новую опцию

Выбор этой опции контекстного меню эквивалентен выбору команды Окно⇨Скрыть. Указанная команда имеет свойство ID, установленное в значение 865, что выясняется после выполнения следующего оператора.

```
MsgBox CommandBars("Worksheet Menu Bar") _
  .Controls("Окно").Controls("Скрыть").ID
```

Удаление опций из контекстного меню

Представленная ниже процедура использует метод Delete для удаления опции, добавленной с помощью процедуры предыдущего раздела.

```
Sub RemoveItemFromShortcut()
    On Error Resume Next
    CommandBars("Cell").Controls("Перенос слов").Delete
End Sub
```

Оператор On Error Resume Next приводит к игнорированию сообщения об ошибке, которое генерируется в том случае, если опции меню не существует.

Следующая процедура удаляет опцию Скрыть из двух контекстных меню: одно из них отображается при щелчке правой кнопкой мыши на заголовке строки, а второе — при щелчке правой кнопкой мыши на заголовке столбца.

```
Sub RemoveHideMenuItems()
    CommandBars("Column").Controls("Скрыть").Delete
    CommandBars("Row").Controls("Скрыть").Delete
End Sub
```

Отключение опций контекстного меню

В качестве альтернативы удалению опций меню можно предложить отключить одну или несколько опций контекстного меню, пока приложение выполняет свою задачу. Когда опция меню отключена, она отображается на экране светло-серым цветом, и щелчок на ней не приводит к выполнению назначенного действия. Приведенная далее процедура отключает опцию Скрыть в контекстных меню Row (Строка) и Column (Столбец).

```
Sub DisableHideMenuItems()
    CommandBars("Column").Controls("Скрыть").Enabled = False
    CommandBars("Row").Controls("Скрыть").Enabled = False
End Sub
```

Отключение контекстных меню

Вы также можете отключать целые контекстные меню. Например, при необходимости вы вправе ограничить доступ пользователя к командам, которые обычно отображаются при щелчке правой кнопкой мыши на ячейке. Следующая процедура `DisableCell` отключает контекстное меню Cell (Ячейка). После выполнения процедуры щелчок на ячейке правой кнопкой мыши не приведет к отображению контекстного меню.

```
Sub DisableCell()
    CommandBars("Cell").Enabled = False
End Sub
```

Если необходимо отключить *все* контекстные меню, воспользуйтесь следующей процедурой.

```
Sub DisableAllShortcutMenus()
    Dim cb As CommandBar
    For Each cb In CommandBars
        If cb.Type = msoBarTypePopup Then _
          cb.Enabled = False
    Next cb
End Sub
```

НА ЗАМЕТКУ: После отключения всех контекстных меню полученное состояние интерфейса сохраняется между сеансами работы. Таким образом, необходимо восстановить доступ к контекстным меню перед закрытием Excel. Для этого измените предыдущую процедуру, чтобы она устанавливала свойство `Enabled` в значение `True`.

Сброс контекстных меню

Метод `Reset` восстанавливает первоначальное состояние контекстного меню (состояние по умолчанию). Если приложение добавляет опции в контекстное меню, то лучше удалять их по отдельности перед закрытием приложения. В противном случае следует удалить изменения в меню, которые внесены другими приложениями.

Следующая процедура восстанавливает контекстное меню Cell (Ячейка) в исходное состояние.

```
Sub ResetCellMenu()
    CommandBars("Cell").Reset
End Sub
```

Создание нового контекстного меню

Существует возможность создания полностью нового контекстного меню. При выполнении кода листинга 23.7 создается контекстное меню, которое называется `MyShortcut` и содержит шесть опций. Для этих опций свойство `OnAction` настроено так, что они приводят к запуску простой процедуры, отображающей одну из вкладок диалогового окна Формат ячеек. Например, процедура `ShowNumberFormat` выглядит следующим образом.

Листинг 23.7. Создание полностью нового контекстного меню

```
Sub CreateShortcut()
    Dim myBar As CommandBar
    Dim myItem As CommandBarControl

    DeleteShortcut
    Set myBar = CommandBars.Add _
      (Name:="MyShortcut", Position:=msoBarPopup, Temporary:=True)

    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Числовой формат..."
        .OnAction = "ShowFormatNumber"
        .FaceId = 1554
    End With

    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Выравнивание..."
        .OnAction = "ShowFormatAlignment"
        .FaceId = 217
    End With

    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Шрифт..."
        .OnAction = "ShowFormatFont"
        .FaceId = 291
    End With

    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Границы..."
        .OnAction = "ShowFormatBorder"
        .FaceId = 149
        .BeginGroup = True
    End With

    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Узор..."
        .OnAction = "ShowFormatPatterns"
        .FaceId = 1550
    End With

    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Защита..."
        .OnAction = "ShowFormatProtection"
        .FaceId = 2654
    End With
End Sub
```

На рис. 23.11 показано созданное с помощью кода контекстное меню.

Созданное контекстное меню вы сможете отобразить, воспользовавшись методом `ShowPopup`. Следующая процедура, которая расположена в модуле кода объекта `ЭтаКнига`, выполняется каждый раз, когда пользователь щелкает правой кнопкой мыши на ячейке.

```
Private Sub Worksheet_BeforeRightClick(ByVal Target _
  As Excel.Range, Cancel As Boolean)
    If Union(Target.Range("A1"), Range("data")).Address = _
      Range("data").Address Then
        CommandBars("MyShortcut").ShowPopup
        Cancel = True
    End If
End Sub
```

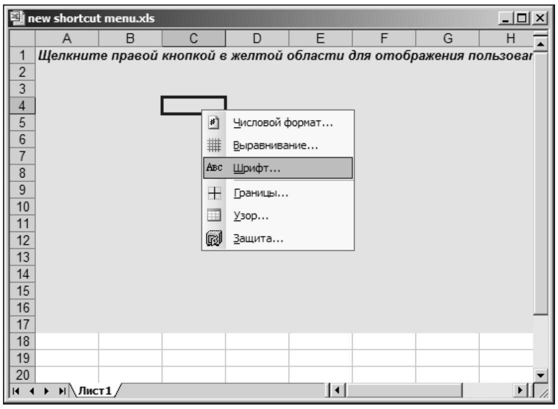


Рис. 23.11. Это контекстное меню создано с помощью кода VBA

Если ячейка, на которой пользователь щелкнул правой кнопкой мыши, находится в пределах именованного диапазона данных, появляется контекстное меню `MyShortcut`. Установка аргумента `Cancel` в значение `True` позволяет удостовериться, что обычно контекстное меню не отображается.

НА КОМПАКТ-ДИСКЕ

На прилагаемом к книге компакт-диске содержится пример создания нового контекстного меню и отображения его вместо обычного контекстного меню Cell (Ячейки).

Глава 24

Предоставление справки в приложениях

В ЭТОЙ ГЛАВЕ...

В этой главе освещаются вопросы, касающиеся справочной информации пользовательских приложений Excel. Изучаемая программа предоставляет большое количество средств для составления справочных руководств.

- Основное назначение справочного руководства к приложению.
- Как создать справочное руководство, используя только средства Excel.
- Принципы создания файлов справочного руководства с помощью справочной системы Windows или справочной системы HTML.
- Как связать файлы справочного руководства с приложением.
- Использование помощника по Office для предоставления справки.
- Другие способы отображения справочной информации.

Пользователи компьютеров со временем становятся избалованными. На ранних этапах существования персональных компьютеров компании-производители программного обеспечения редко добавляли к своему продукту интерактивное справочное руководство. А предоставляемая "справка" была, как правило, бесполезной. Теперь практически каждая программа имеет справочное руководство. Толстые бумажные руководства по использованию программного обеспечения уже можно заносить в "Красную книгу" как вымирающий вид (туда им и дорога)!

Справка в приложениях Excel

Если в Excel разрабатывается довольно сложное приложение, то может возникнуть необходимость в создании справочного руководства для потенциальных пользователей приложения. Таким образом, пользователям будет комфортно работать с приложением, и они не станут беспокоить вас телефонными звонками. Еще одним преимуществом интерактивного справочного руководства является постоянная его доступность (это руководство невозможно потерять или "похоронить" под огромной стопкой книг).

Справочное руководство для пользователей можно добавить к приложению несколькими способами, различающимися по степени сложности. Метод, который будет вами избран, зависит от масштаба и сложности конкретного приложения, а также от того, сколько времени вы можете себе позволить потратить на создание справочного руководства для приложения. Некоторые приложения нуждаются исключительно в простом наборе инструкций по запуску и управлению. Для других приложений необходимо создавать полноценную справочную систему с поисковым средством. Но для большинства приложений требуется справочное руководство среднего уровня — достаточно сложное для создания без подготовки, но простое для изучения.

Диалоговая справка

В последнее время каждая программа оснащается тем, что мы называем диалоговой справочной системой. Как того и следовало ожидать, этим термином называется справочная информация, размещенная в Internet. Но некоторые пользователи до сих пор заблуждаются, полагая что все справочные данные по использованию программы хранятся на жестком диске.

Чтобы разграничить две справочные системы, мы будем использовать следующее соглашение. Пусть термин “справочная система” обозначает информацию, предоставляемую приложением (т.е. сохраненную на жестком диске). Под “диалоговой справочной системой” будем понимать исключительно данные, запрашиваемые из Internet. В Excel 2003 справочные сведения впервые по умолчанию загружаются из Internet, а не с жесткого диска.

О примерах, приводимых в этой главе

В данной главе используется простое приложение рабочей книги, которое демонстрирует различные способы предоставления справочной информации. Приложение использует данные, которые хранятся на рабочем листе, для создания и печати писем.

Как можно заметить из рисунка, ячейки с числами отображают общее количество записей в базе данных (`C2`, рассчитывается с помощью формулы), номер текущей записи (`C3`), первая запись, которая будет напечатана (`C4`) и последняя запись, которая будет напечатана (`C5`). Для того чтобы отобразить определенную запись, пользователь вводит значение в ячейку `C3`. Распечатать последовательность писем можно, указав первую и последнюю запись в полях `C4` и `C5`.

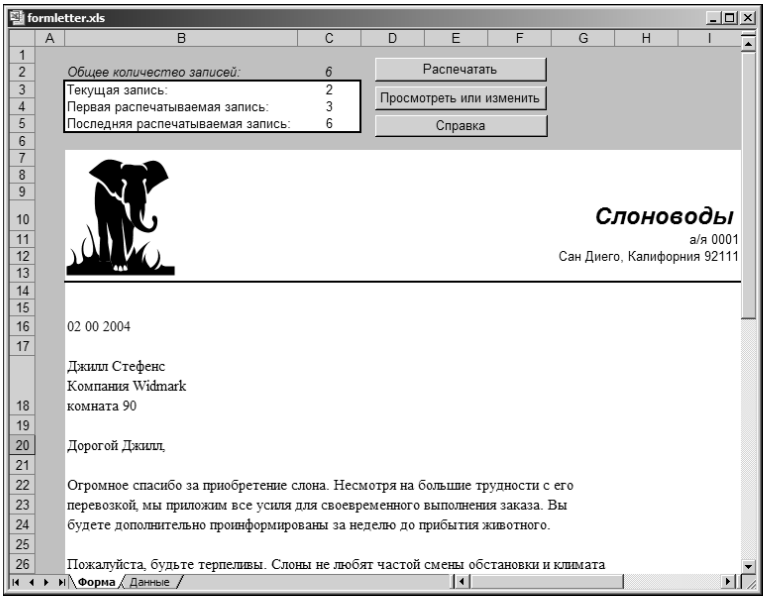
Приложение имеет очень простую структуру. Оно состоит из нескольких отдельных компонентов, которые поддерживают различные способы отображения справочной информации.

Рабочая книга состоит из следующих компонентов.

`Форма` — рабочий лист, который содержит текст приложения.

`Данные` — рабочий лист, который содержит базу данных из семи полей.

`Справка` — этот рабочий лист представлен только в примерах, в которых текст справочной системы сохраняется в рабочих книгах.



`PrintMod` — модуль VBA, который содержит макросы печати писем.

`HelpMod` — модуль VBA, который содержит макросы, отображающие текст справочной системы. Содержимое этого модуля может меняться в зависимости от контекста.

`UserForm1` — пользовательское диалоговое окно, отображаемое только тогда, когда избранная методика предоставления справочных сведений предполагает использование форм `UserForm`.

В настоящей главе интерактивные справочные руководства будут разделены на две категории.

- *Неофициальные справочные системы.* Этот метод отображения справочной информации требует использования только стандартных компонентов Excel (например, диалоговых окно `UserForm`).
- *Официальные справочные системы.* Такие справочные системы используют или откомпилированный файл HLP, который создается с помощью средства Windows Help System, или откомпилированный файл CHM, который создается с помощью средства HTML Help System.

Создание откомпилированного файла справочной системы является непростой задачей, но если приложение достаточно сложное, получение такого файла обязательно оправдает затраченные усилия. Создавать такой файл целесообразно только в том случае, когда его будет использовать большое количество людей.

НА КОМПАКТ-ДИСКЕ

Все примеры из этой главы вы найдете на прилагаемом к книге компакт-диске.

Справочная система, построенная с помощью компонентов Excel

Возможно, самым простым методом предоставления справочного руководства является использование средств, которые поддерживаются в Excel. Основное преимущество данного метода — необязательность изучения методов создания файлов WinHelp или HTML Help. Ознакомление с последними методами может занять довольно много времени и увеличить срок разработки приложения.

В данном разделе приведен обзор методик предоставления справочной информации, в которых используются следующие встроенные в Excel компоненты.

- *Комментарии к ячейкам.* Проще не бывает!
- *Элемент управления Text Box (Текстовое поле).* Простой макрос, который отображает на экране текстовое поле, содержащее справочную информацию.
- *Рабочий лист.* Простым способом добавления справочного руководства в приложение является вставка рабочего листа, на котором отображается необходимая справочная информация. После ввода информации рабочий лист можно назвать Справка. Если пользователь щелкнет на ярлыке этого листа, будет активизирована справочная система приложения.
- *Пользовательское диалоговое окно.* Некоторые методы построения справочного руководства подразумевают использование диалоговых окон `UserForm`.

Использование комментариев к ячейкам для предоставления справки

Самый простой способ предоставления справочной информации конечному пользователю заключается в добавлении комментариев к ячейкам. Эта методика является наиболее подходящей для описания типа вводимых данных, которые принимаются в определенной ячейке. Когда пользователь наводит указатель мыши на ячейку, содержащую комментарий, текст комментария отображается в небольшом окне. Еще одним преимуществом этой методики является отсутствие необходимости в применении макросов.

Помимо этого, возможно автоматическое отображение комментариев к ячейкам. Следующий оператор VBA обеспечивает отображение индикаторов комментариев в ячейках.

```
Application.DisplayCommentIndicator = xlCommentIndicatorOnly
```

СОВЕТ Вы также можете воспользоваться командой Excel Данные⇨Проверка, которая отображает диалоговое окно и позволяет указать критерии проверки введенных в ячейку или диапазон данных. Вкладка Сообщение для ввода диалогового окна Проверка вводимых значений позволяет указать текст сообщения, которое будет отображаться при активизации ячейки. Этот текст ограничен длиной в 250 символов.

Применение текстового поля для предоставления справки

Использование элемента управления `Text Box` для отображения справочной информации является еще одним простым в реализации методом предоставления справочной информации пользователю. Создать элемент управления `Text Box` достаточно просто. Используя кнопку Надпись на панели инструментов Рисование, введите необходимую справочную информацию и отформатируйте текст по своему вкусу. На рис. 24.1 показан пример элемента управления `Text Box`, который применяется для отображения справочной информации.

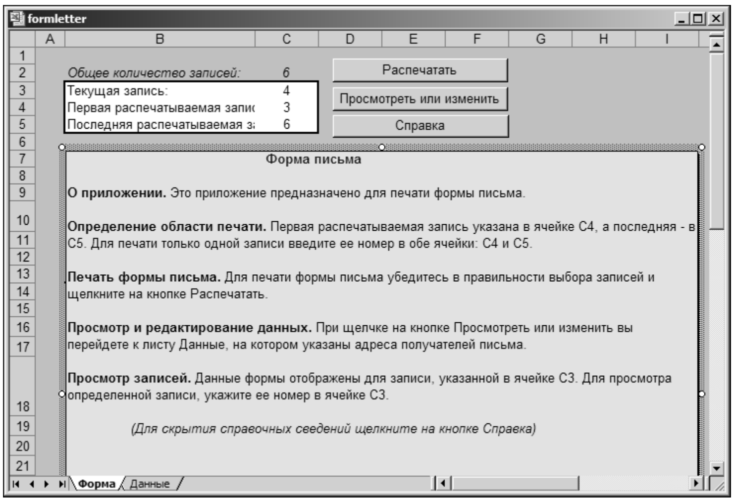


Рис. 24.1. Использование элемента управления `Text Box` для отображения справочного руководства пользователя

Использование кнопки Надпись на панели инструментов Рисование является более предпочтительным способом, чем применение элемента управления ActiveX Поле на панели инструментов Элементы управления, так как первый элемент управления предоставляет возможность форматирования текста. Другими словами, элемент управления Надпись на панели инструментов Рисование позволяет форматировать каждый символ текста в элементе управления.

Зачастую требуется, чтобы элемент управления `Text Box` оставался невидимым. Для этого требуется создать для приложения макрос и кнопку, которые будут устанавливать свойство `Visible` элемента управления `Text Box`. Примером такого макроса может служить следующий код. В данном случае элемент управления `Text Box` называется `HelpText`.

```
Sub ToggleHelp()
    ActiveSheet.TextBoxes("HelpText").Visible = _
      Not ActiveSheet.TextBoxes("HelpText").Visible
End Sub
```

Использование рабочего листа для сохранения текста справочного руководства

Еще одним способом добавления справочного руководства к приложению является использование макроса, который активизирует отдельный рабочий лист, содержащий текст справочного руководства. Достаточно связать макрос с кнопкой на панели инструментов, элементом управления или опцией меню, чтобы при необходимости получить доступ к справочному руководству.

На рис. 24.2 показан пример рабочего листа, в котором находится справочная информация. Мною создан диапазон, который содержит текст справки и имитирует страницу блокнота (такой дизайнерский прием некоторым может нравиться, а некоторым — нет).

Для того чтобы не позволить пользователю прокручивать рабочий лист `Справка`, макрос изменяет свойство `ScrollArea` рабочего листа. Таким образом, значение этого свойства не сохраняется вместе с рабочей книгой, его необходимо устанавливать каждый раз при активизации рабочего листа. Кроме того, рабочий лист защищен, чтобы пользователь не мог вносить изменения в отображаемый текст. Более того, первая строка зафиксирована, чтобы кнопка Назад всегда оставалась видимой, независимо от того, насколько прокручивается рабочий лист.

Основным недостатком использования этой методики можно считать невозможность отображения текста справки одновременно с основной рабочей областью. Одним из возможных решений является применение макроса, который будет открывать новое окно для отображения рабочего листа, содержащего текст справочного руководства.

Отображение справочной информации в пользовательском диалоговом окне

Самым удачным способом предоставления справочного руководства средствами Excel можно считать использование диалоговых окон `UserForm`, в которых отображаются справочные сведения. В данном разделе описываются некоторые методики, которые подразумевают применение пользовательских диалоговых окон.

ИСПОЛЬЗОВАНИЕ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ LABEL ДЛЯ ОТОБРАЖЕНИЯ ТЕКСТА СПРАВОЧНОГО РУКОВОДСТВА

На рис. 24.3 показано диалоговое окно `UserForm`, которое содержит два элемента управления `Label`: один элемент управления используется для отображения заголовка, а второй — для отображения самого текста. Элемент управления `SpinButton` позволяет пользователю перемещаться по разделам справочного руководства. Сам текст сохраняется на рабочем листе. Названия разделов справочного руководства хранятся в столбце `A`, а текст разделов — в столбце `B`.

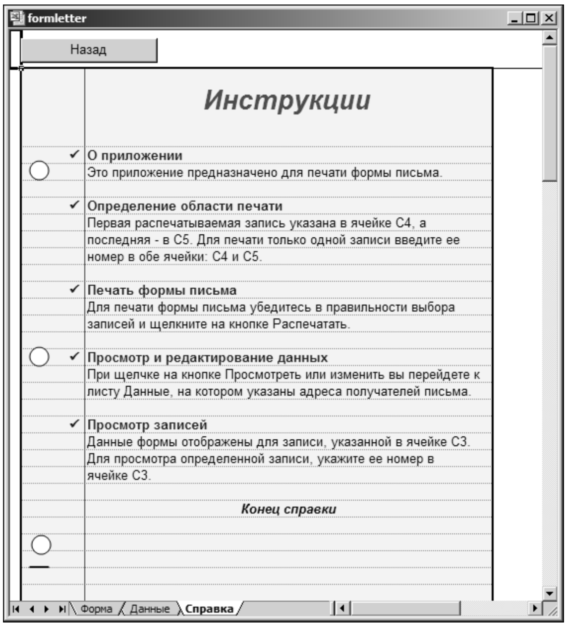


Рис. 24.2. Размещение справочного руководства на отдельном рабочем листе является одним из самых простых способов

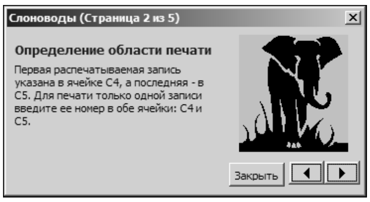


Рис. 24.3. Щелчок на элементе управления `SpinButton` приведет к отображению другой темы справочного руководства

Щелчок на элементе управления `SpinButton` приводит к выполнению представленной ниже процедуры. Эта процедура устанавливает свойство `Caption` двух элементов управления `Label` так, чтобы они отображали текст из соответствующей строки рабочего листа `Справка`.

```
Private Sub SpinButton1_Change()
    HelpTopic = SpinButton1.Value
    LabelTopic.Caption = Sheets("Справка"). _
      Cells(HelpTopic, 1)
```

```
    LabelText.Caption = Sheets("Справка").Cells(HelpTopic, 2)
    Me.Caption = APPNAME & ": Topic " & HelpTopic & "/" _
      & SpinButton1.Max
End Sub
```

В данном случае APPNAME является глобальной константой, которая содержит название приложения.

ИСПОЛЬЗОВАНИЕ "ПРОКРУЧИВАЕМОГО" ЭЛЕМЕНТА УПРАВЛЕНИЯ LABEL ДЛЯ ОТОБРАЖЕНИЯ ТЕКСТА СПРАВОЧНОГО РУКОВОДСТВА

Данная методика заключается в отображении текста справочного руководства с помощью единственного элемента управления Label. По причине того, что элемент управления Label не может содержать вертикальной полосы прокрутки, он размещается внутри элемента управления Frame, поддерживающего использование обеих полос прокрутки. На рис. 24.4 показан пример применения диалогового окна UserForm, настроенного описанным выше способом. Пользователь может прокручивать текст с помощью полосы прокрутки элемента управления Frame.

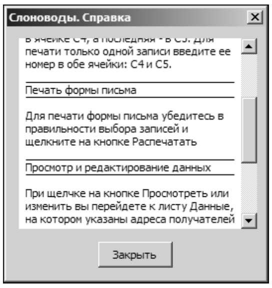


Рис. 24.4. Использование элемента управления Frame *с целью добавления в элемент управления* Label *полос прокрутки*

Текст, который отображается в элементе управления Label, получен из рабочего листа Справка. Передача текста осуществляется на этапе инициализации диалогового окна UserForm. Листинг 24.1 содержит исходный код процедуры UserForm_Initialize этого диалогового окна. Обратите внимание на то, как код меняет значение свойства ScrollHeight элемента управления Frame с целью обеспечить прокрутку для всего элемента управления Label. В данном случае APPNAME является глобальной константой, которая содержит название приложения.

Листинг 24.1. Создание элемента управления Label с прокруткой, который будет отображать текст, содержащийся на рабочем листе

```
Private Sub UserForm_Initialize()
    Me.Caption = APPNAME & ". Справка"
    LastRow = Sheets("Справка").Range("A65536").End(xlUp).Row
    txt = ""
    For r = 1 To LastRow
    txt = txt & Sheets("Справка").Cells(r, 1).Text & vbCrLf

    Next r
    With Label1
        .Top = 0
        .Caption = txt
        .Width = 160
        .AutoSize = True
    End With
    Frame1.ScrollHeight = Label1.Height
    Frame1.ScrollTop = 0
End Sub
```

По причине того, что элемент управления Label не может отображать форматированный текст, в рабочем листе Справка использовалось подчеркивание для выделения названий разделов справочного руководства.

ИСПОЛЬЗОВАНИЕ РАСКРЫВАЮЩЕГОСЯ СПИСКА ДЛЯ ВЫБОРА РАЗДЕЛА СПРАВОЧНОГО РУКОВОДСТВА

Пример, приведенный в этом разделе, расширит возможности предыдущего примера. На рис. 24.5 показано диалоговое окно UserForm, которое содержит элемент управления ComboBox и элемент управления Label. Пользователь может выбрать раздел справочного руководства из раскрывающегося списка, который предоставляется элементом управления ComboBox, а также последовательно просмотреть разделы с помощью кнопок Назад и Далее.

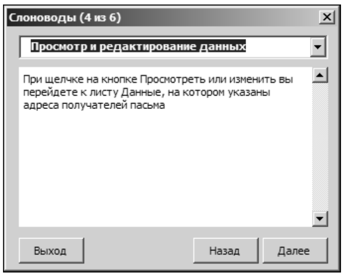


Рис. 24.5. Выбор текста, отображаемого в элементе управления Label, с помощью раскрывающегося списка

Приведенный пример (листинг 24.2) немного сложнее, чем пример из предыдущего раздела, но он предоставляет большую гибкость. В данном случае используется методика “Label Frame” (она рассматривалась ранее), что позволяет вводить разделы справочного руководства любой длины.

Текст справочного руководства хранится в рабочем листе, который называется Справка. Он расположен в двух столбцах: A и B. Первый столбец содержит названия разделов справочного руководства, а второй — текст разделов. Опции элемента управления ComboBox добавляются во время выполнения процедуры UserForm_Initialize. Переменная CurrentTopic определена на уровне модуля и содержит целое число, которое идентифицирует выбранный раздел справочного руководства.

Листинг 24.2. Метод “Label Frame”

```
Private Sub UpdateForm()
    ComboBoxTopics.ListIndex = CurrentTopic - 1
    Me.Caption = HelpFormCaption & _
      " (" & CurrentTopic & " из " & TopicCount & ")"

    With LabelText
        .Caption = HelpSheet.Cells(CurrentTopic, 2)
        .AutoSize = False
        .Width = 212
        .AutoSize = True
    End With
    With Frame1
        .ScrollHeight = LabelText.Height + 5
        .ScrollTop = 1
    End With

    If CurrentTopic = 1 Then
        NextButton.SetFocus
```

```
    ElseIf CurrentTopic = TopicCount Then
        PreviousButton.SetFocus
    End If
    PreviousButton.Enabled = CurrentTopic <> 1
    NextButton.Enabled = CurrentTopic <> TopicCount
End Sub
```

Использование помощника по Office для отображения справочного руководства

Скорее всего, вы уже знакомы с помощником по Office — смешным анимированным персонажем, который всегда готов оказать помощь пользователю. Многие пользователи не признают такой способ предоставления справочной информации. Судя по всему, Microsoft осознала свою ошибку, и в Office 2003 по умолчанию он отключен.



В Excel 2003 помощник по умолчанию не устанавливается. Поэтому данный метод получения справки далеко не самый лучший.

Однако помощник по Office предоставляет определенный программный интерфейс, поэтому (если необходимо) его можно использовать для отображения справочной информации собственного приложения. На рис. 24.6 показан помощник по Office, который представляет текст раздела справочного руководства.

Основной процедурой использования помощника по Office для отображения текста справочного руководства является код, показанный в листинге 24.3. Текст справочного руководства хранится на рабочем листе в двух столбцах. Этот рабочий лист называется `Справка`. Столбец `A` содержит названия разделов, а столбец `B` — текст самих разделов.

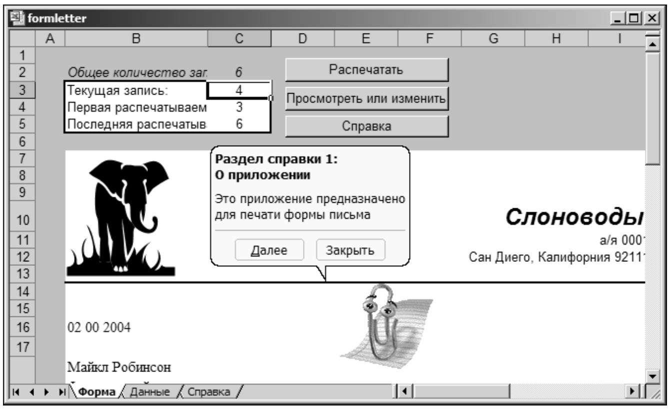


Рис. 24.6. Использование помощника по Office для предоставления собственной справочной информации

Листинг 24.3. Вызов помощника по Office для отображения справочной информации

```
Public Const APPNAME As String = "Слоноводы"
Dim Topic
Dim HelpSheet

Sub ShowHelp()
    Set HelpSheet = ThisWorkbook.Worksheets("Справка")
```

```
    Application.Assistant.On = True
    Topic = 1
    With Assistant.NewBalloon
        .Heading = "Раздел справки " & Topic & ": " & vbCrLf & Help-
Sheet.Cells(Topic, 1)
        .Text = HelpSheet.Cells(Topic, 2)

        .Button = msoButtonSetNextClose
        .BalloonType = msoBalloonTypeButtons
        .Mode = msoModeModeless
        .Callback = "ProcessRequest"
        .Show
    End With

End Sub
```

Процедура начинает свою работу с проверки работоспособности помощника по Office. После этого создается новый объект `Balloon` (помощник по Office отображает текст справочного руководства во всплывающем окне) и использует первый раздел справочного руководства из рабочего листа `Справка` для задания свойств `Heading` и `Text` этого объекта. Свойства объекта `Balloon` устанавливаются таким образом, чтобы он содержал кнопки Далее и Закрыть, как это по умолчанию определено в мастере. Затем процедура устанавливает свойство `Mode` в значение `msoModeModeless`, что позволяет пользователю работать над другими приложениями в процессе изучения справочной информации. Свойство `Callback` содержит имя процедуры, которая будет выполняться при щелчке на кнопке. Наконец, всплывающее окно помощника отображается с помощью метода `Show`.

Процедура `ProcessRequest`, которая показана в листинге 24.4, вызывается каждый раз, когда пользователь щелкает на одной из кнопок всплывающего окна.

Листинг 24.4. Предоставление собственного справочного руководства с использованием помощника по Office

```
Sub ProcessRequest(bln As Balloon, lbtn As Long, lPriv As Long)
    Dim NumTopics As Integer

    NumTopics = Application.WorksheetFunction.CountA(HelpSheet.Range("A:A"))
    Assistant.Animation = msoAnimationCharacterSuccessMajor
    Select Case lbtn
    Case msoBalloonButtonBack
        If Topic <> 1 Then Topic = Topic - 1
    Case msoBalloonButtonNext
        If Topic <> NumTopics Then Topic = Topic + 1
    Case msoBalloonButtonClose
        bln.Close
        Exit Sub
    End Select
    With bln
        .Close
        Select Case Topic
            Case 1:  .Button = msoButtonSetNextClose
            Case NumTopics: .Button = msoButtonSetBackClose
            Case Else: .Button = msoButtonSetBackNextClose
        End Select
        .Heading = "Раздел справки " & Topic & ": " & vbCrLf & _
HelpSheet.Cells(Topic, 1)
        .Text = HelpSheet.Cells(Topic, 2)
        .Show
    End With
End Sub
```

Процедура ProcessRequest отображает один из анимационных персонажей, после чего использует конструкцию Select Case для выполнения определенного действия, которое зависит от того, на какой из кнопок щелкнул пользователь. Информация о кнопке, на которой щелкнул пользователь, передается в процедуру с помощью переменной lbtn. Кроме того, процедура указывает, какие кнопки необходимо отображать, что зависит от текущего раздела справочного руководства.

Если вам необходимо получить дополнительную информацию о программном интерфейсе помощника по Office, то стоит обратиться к интерактивному справочному руководству.

Этот пример находится на прилагаемом к книге компакт-диске.

Имитация средства Что это такое?

Отдельные диалоговые окна Excel содержат в строке заголовка маленькую кнопочку, рядом с кнопкой закрытия окна. На ней изображен вопросительный знак. В Excel 2003 щелчок на этой кнопке приводит к отображению справочных сведений о текущем диалоговом окне. В предыдущих версиях программы щелчок на этой кнопке активизировал инструмент Что это такое? Пользователь мог щелкнуть этим инструментом на любом элементе управления окна, чтобы отобразить справочные сведения о нем.

Пользовательские формы имеют все необходимые средства для имитации этого инструмента. Но перед вами стоит парадоксальная и неразрешимая проблема — это средство не работает. Например, пользовательская форма имеет свойства WhatThisButton и WhatThisHelp. Тем не менее, они не позволяют отобразить справочные сведения. Детально о предоставлении справки с помощью HTML рассказано в следующем разделе.

Каждый объект UserForm имеет свойство ControlTipText. При определении текста для этого свойства он будет отображаться в результате наведения указателя мыши на элемент управления.

В качестве альтернативного варианта можно воспользоваться свойством MouseMove для отображения текста в отдельном поле в зависимости от расположения указателя мыши. На рис. 24.7 показан пример этого метода. Текст отображается с помощью элемента управления Label.

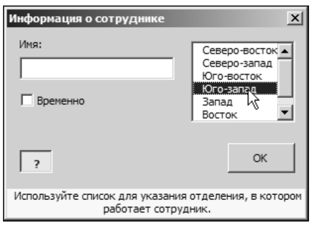


Рис. 24.7. Отображение справочных сведений в элементе управления Label

Использование средства HTML Help System

В настоящее время в приложениях Windows чаще всего используются справочные руководства на основе технологии HTML Help, с помощью которой можно управлять файлами справочной системы в формате CHM. Совсем недавно эта технология заменила старую технологию WinHelp с файлами формата HLP. Многие новые приложения, которые создаются компанией Microsoft, для предоставления справочной информации используют исключительно технологию HTML Help.

В данном разделе кратко рассмотрена только справочная система HTML Help. Способы создания сложных справочных руководств выходят за пределы тем, рассматриваемых в этой книге.

Если вы планируете разрабатывать полномасштабные справочные руководства, то рекомендуется приобрести специальное программное обеспечение, которое предназначено для выполнения этой задачи. Программное обеспечение создания справочных руководств облегчает редактирование таких документов, так как выполняет большую часть рутинных операций. На данный момент существует немало бесплатных, условно-бесплатных, а также коммерческих программ. Возможно, самым популярным приложением в этой области является RoboHELP компании eHelp Corporation. RoboHELP позволяет создавать файлы в формате обеих систем (WinHelp и HTML Help). Для получения дополнительной информации об этой программе посетите Web-узел по адресу `http://www.ehelp.com`.

Как отмечалось ранее, Microsoft разработала технологию HTML Help в качестве нового стандарта систем создания интерактивных справочных руководств. Эта система компилирует несколько файлов HTML в компактное справочное руководство. Система HTML Help может использовать дополнительные средства, например, графические файлы, элементы управления ActiveX, сценарии, а также документы DHTML (Dynamic HTML). На рис. 24.8 показан пример справочной системы, созданной по этой технологии (к сожалению, на английском языке — *прим. ред*).

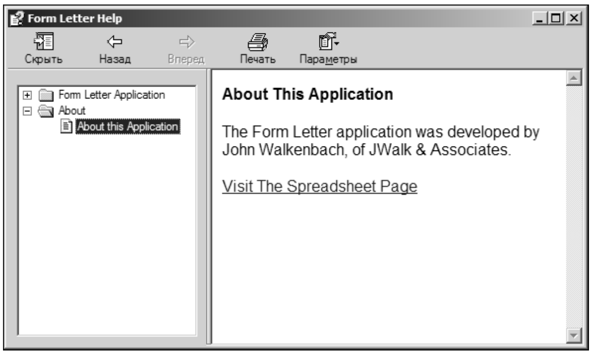


Рис. 24.8. Пример справочной системы HTML Help

Система HTML Help использует для отображения данных программу HTML Help Viewer, которая создана на ядре браузера Internet Explorer. Информация отображается в основном окне, а содержание, указатель и инструменты поиска — на отдельной панели. Кроме того, текст справочного руководства может содержать стандартные гиперссылки, которые указывают на другие разделы справочного руководства или даже на другие узлы в Internet. Важно также, что технология HTML Help получает доступ к файлам, которые хранятся на Web-сервере. Это идеальное средство для предоставления пользователям свежей информации, которой не существовало на момент создания справочного руководства.

Вы должны были заметить, что справочная система Excel 2003 существенно отличается от используемых в ранних версиях программы. Это вызвано, в первую очередь, ее интеграцией с областью задач. Однако следует отметить, что отдельное окно справочная система в Excel 2003 тоже имеет. На мой взгляд, подобное излишество только усложняет задачи пользователя, связанные с поиском необходимых данных.

Как и в случае WinHelp, вам необходимо использовать специальный компилятор, который создает файлы, понятные HTML Help. Описание программы HTML Help Workshop, а также большое количество дополнительных данных приводится на Web-узле компании Microsoft по адресу: `http://msdn.microsoft.com/library/tools/htmlhelp/chm/HH1Start.htm`.

> **СОВЕТ** Чтобы получить представление о процедуре создания файлов HTML, отобразите одну из тем интерактивного справочного руководства Excel. После этого щелкните правой кнопкой мыши на документе и выберите Вид⇨Источник. Если вы выполнили эту команду, у вас появится возможность просмотреть исходный код документа.

Связывание файлов справочного руководства с приложением

Если вы решили использовать одну из "стандартных" систем поддержки справочного руководства (т.е. WinHelp или HTML Help), то конечный файл справочного руководства можно связать с приложением одним из двух способов: с помощью диалогового окна Project Properties (Свойства проекта) или посредством создания специального кода VBA.

В редакторе VBE выберите Tools⇨xxx Properties (Сервис⇨Свойства xxx) (где xxx соответствует имени проекта). В диалоговом окне Project Properties (Свойства проекта) перейдите на вкладку General (Общие) и укажите файл справочного руководства, который будет использоваться данным приложением (это может быть или файл CHM, или HLP).

Хорошей практикой является хранение файлов справочного руководства в той же папке, в которой располагается само приложение. Представленный далее оператор связывает приложение с файлом `Myfuncs.chm`. Этот файл находится в той же папке, что и файл рабочей книги.

```
ThisWorkbook.VBProject.HelpFile = _
  ThisWorkbook.Path & "\Myfuncs.chm"
```

После связывания файла справочного руководства с приложением можно отобразить определенные справочные сведения в следующих ситуациях.

- Когда пользователь нажимает клавишу <F1> при выборе пользовательской функции в диалоговом окне Мастер функций.
- Когда пользователь нажимает клавишу <F1> при отображенном диалоговом окне `UserForm`. В результате отображается раздел справочного руководства, который связан с элементом управления, активным в момент нажатия клавиши <F1>.

СВЯЗЫВАНИЕ РАЗДЕЛА СПРАВОЧНОГО РУКОВОДСТВА С ФУНКЦИЕЙ VBA

Если с помощью кода VBA создаются новые функции рабочего листа, то может возникнуть необходимость связать файл справочного руководства и идентификатор раздела с каждой функцией. Как только связь между этими элементами будет установлена, раздел справочного руководства сможет быть отображен после нажатия клавиши <F1> в диалоговом окне Мастер функций.

Для того чтобы указать идентификатор раздела для новой функции рабочего листа, следуйте приведенным ниже инструкциям.

1. Создайте функцию.
2. Удостоверьтесь, что с проектом связан файл справочного руководства (для получения дополнительной информации обратитесь к предыдущему разделу).

3. В редакторе VBE нажмите клавишу <F2> для активизации окна Object Browser.
4. Укажите проект в раскрывающемся списке Project/Library.
5. В списке Classes выберите модуль, который содержит функцию.
6. В окне Member (Компонент) укажите функцию.
7. Щелкните на функции правой кнопкой мыши и выберите Properties из появившегося контекстного меню. Таким образом, будет отображено диалоговое окно Member Options (Свойства компонента), как показано на рис. 24.9.

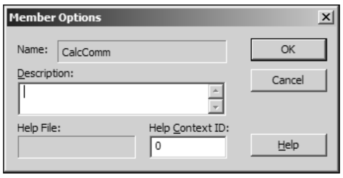


Рис. 24.9. Определите идентификатор для пользовательской функции

8. Введите идентификатор раздела справочного руководства, который связан с этой функцией. Вы также можете добавить описание функции.



Диалоговое окно Member Options не позволяет указать новый файл справочного руководства. Это диалоговое окно всегда использует файл, который ранее связан с проектом.

Может оказаться, что проще создать код VBA, который будет настраивать идентификатор раздела и файл справочного руководства для новых функций. Для этого воспользуйтесь методом `MacroOptions`. Следующая процедура использует метод `MacroOptions` для определения описания функции, применяемого файла справочного руководства и идентификатора раздела для двух функций (`AddTwo` и `Squared`).

```
Sub SetOptions()
'   Установить параметры для функции AddTwo
    Application.MacroOptions Macro:="AddTwo", _
        Description:="Возвращает сумму двух чисел", _
        HelpFile:=ThisWorkbook.Path & "\Myfuncs.hlp", _
        HelpContextID:=1000

'   Установить параметры для функции Squared
    Application.MacroOptions Macro:="Squared", _
        Description:="Возвращает квадрат аргумента", _
        HelpFile:=ThisWorkbook.Path & "\Myfuncs.hlp", _
        HelpContextID:=2000
End Sub
```

Другие способы отображения справочного руководства в формате WinHelp или HTML Help

VBA предоставляет несколько дополнительных способов отображения разделов справочного руководства. Эти способы рассматриваются в следующих разделах.

Использование метода Help

Воспользуйтесь методом `Help` объекта `Application` для отображения файла справочного руководства. Этот файл может иметь как формат WinHelp HLP, так и формат HTML Help CHM. Данный метод работает даже тогда, когда для файла не определены идентификаторы раздела.

Метод `Help` имеет следующий синтаксис.

```
Application.Help(helpFile, helpContextID)
```

Оба аргумента являются необязательными. Если имя файла справочного руководства не указано, то будет отображаться файл справочного руководства Excel. Если опустить идентификатор раздела, то указанный файл будет отображен с использованием раздела, принятого по умолчанию.

Следующий пример отображает раздел файла `Myapp.hlp`, принятый по умолчанию. Файл должен находиться в той же папке, что и рабочая книга приложения, из которого он вызывается. Обратите внимание: второй аргумент не указан.

```
Sub ShowHelpContents()
    Application.Help ThisWorkbook.Path & "\Myapp.hlp"
End Sub
```

Представленный далее оператор отображает раздел справочного руководства с идентификатором 1002. При этом используется файл справочного руководства в формате HTML Help, который называется `Myapp.chm`.

```
Application.Help ThisWorkbook.Path & "\Myapp.chm", 1002
```

Отображение справочной информации в окне сообщения

При использовании функции VBA `MsgBox` для отображения окна сообщения можно отобразить кнопку Help (Справка), применив константу `vbMsgBoxHelpButton` в качестве второго аргумента функции `MsgBox`. Кроме того, необходимо в качестве четвертого аргумента указать имя файла справочного руководства. Пятый аргумент (который не обязателен) используется с целью задать идентификатор раздела. Следующий код демонстрирует пример создания окна сообщения, показанного на рис. 24.10.

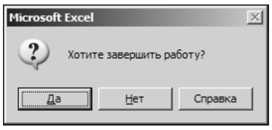


Рис. 24.10. Окно сообщения с кнопкой Справка

```
Sub MsgBoxHelp()
    Msg = "Хотите завершить работу?"
    Buttons = vbQuestion + vbYesNo + vbMsgBoxHelpButton
    HelpFile = ThisWorkbook.Path & "\AppHelp.hlp"
    ContextID = 1002
    Ans = MsgBox(Msg, Buttons, , HelpFile, ContextID)
    If Ans = vbYes Then Call CloseDown
End Sub
```

Отображение справочной информации в окне ввода данных

Функция VBA `InputBox` может отображать кнопку Help (Справка), если шестой аргумент этой функции содержит имя файла справочного руководства. В приведенном далее примере создается окно ввода данных, которое показано на рис. 24.11.

```
Sub ShowInputBox()
    Msg = "Введите значение"
    DefaultVal = 0
    HFile = ThisWorkbook.Path & "\AppHelp.hlp"
    ContextID = 1002
    x = InputBox( _
        Prompt:=Msg, _
        Default:=DefaultVal, _
        HelpFile:=HFile, _
        Context:=ContextID)
End Sub
```



Рис. 24.11. Окно ввода данных с кнопкой Help *(Справка)*

Глава 25

Разработка приложений для пользователей

В ЭТОЙ ГЛАВЕ...

В этой главе обобщена вся информация, представленная в предыдущих главах.

- Описание приложений, ориентированных на пользователя.
- Подробное рассмотрение мастера расчета займа, который генерирует таблицу, отображающую расписание погашения займа с фиксированной процентной ставкой.
- Рассмотрение концепций и методик разработки приложений на основе мастера расчета займа.
- Список действий, которые необходимо выполнить при разработке приложения.

В данной главе описывается приложение, ориентированное на конечного пользователя, — мастер расчета займа. Это приложение нашло практическое применение, хотя мы будем использовать его для изучения методов разработки приложений, ориентированных на пользователя.

Определение приложения, ориентированного на пользователя

Приложение, ориентированное на пользователя, — это приложение Excel, которое можно использовать без специальной подготовки. Такие приложения позволяют получить конечный результат даже тем пользователям, которые совершенно ничего не знают об Excel.

Приложение мастера расчета займа, которое рассматривается в этой главе, относится к приложениям, ориентированным на пользователя (оно разрабатывалось специально для того, чтобы помочь пользователям, которые совершенно ничего не знают о функционировании Excel). Ответив на несколько простых вопросов, вы сможете создать полезный и гибкий рабочий лист, содержащий необходимые формулы.

Мастер расчета займа

Приложение мастера расчета займа создает рабочий лист, который содержит расписание погашения займа с фиксированной процентной ставкой. Расписание амортизации представлено в виде информации о ежемесячных выплатах. Таким образом, отображаются сведения о сумме ежемесячных выплат, а также данные о том, какая часть суммы идет на погашение процентов, а какая — на погашение основной суммы займа. Более того, пользователь может ознакомиться и с информацией о новом балансе.

Альтернативой такого приложения можно считать файл шаблона (`*.XLT`) с необходимыми формулами. Далее вы убедитесь, что приложение в форме мастера предоставляет несколько дополнительных преимуществ.

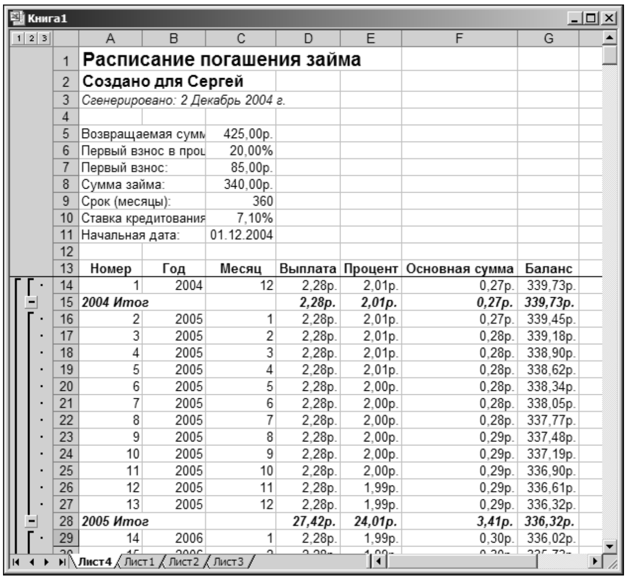
На рис. 25.1 показано расписание амортизации, которое создано приложением мастера расчета займа.

Это приложение доступно на прилагаемом к книге компакт-диске. Оно сохранено в виде незащищенной надстройки Excel.

Использование приложения

Приложение мастера расчета займа отображает последовательность из пяти диалоговых окон, которые запрашивают информацию у пользователя. Как и другие мастера, это приложение позволяет перемещаться вперед и назад по этапам работы мастера. Щелчок на кнопке Готово приведет к созданию нового рабочего листа.

Данное приложение использует единственное диалоговое окно `UserForm` с элементом управления `MultiPage`, которое содержит вкладки для всех пяти этапов работы мастера (рис. 25.2—25.6).



Книга1

	A	B	C	D	E	F	G
1	**Расписание погашения займа**						
2	**Создано для Сергей**						
3	*Сгенерировано: 2 Декабрь 2004 г.*						
4							
5	Возвращаемая сумм		425,00р.				
6	Первый взнос в проц		20,00%				
7	Первый взнос:		85,00р.				
8	Сумма займа:		340,00р.				
9	Срок (месяцы):		360				
10	Ставка кредитования		7,10%				
11	Начальная дата:		01.12.2004				
12							
13	**Номер**	**Год**	**Месяц**	**Выплата**	**Процент**	**Основная сумма**	**Баланс**
14	1	2004	12	2,28р.	2,01р.	0,27р.	339,73р.
15	***2004 Итог***			***2,28р.***	***2,01р.***	***0,27р.***	***339,73р.***
16	2	2005	1	2,28р.	2,01р.	0,27р.	339,45р.
17	3	2005	2	2,28р.	2,01р.	0,28р.	339,18р.
18	4	2005	3	2,28р.	2,01р.	0,28р.	338,90р.
19	5	2005	4	2,28р.	2,01р.	0,28р.	338,62р.
20	6	2005	5	2,28р.	2,00р.	0,28р.	338,34р.
21	7	2005	6	2,28р.	2,00р.	0,28р.	338,05р.
22	8	2005	7	2,28р.	2,00р.	0,28р.	337,77р.
23	9	2005	8	2,28р.	2,00р.	0,29р.	337,48р.
24	10	2005	9	2,28р.	2,00р.	0,29р.	337,19р.
25	11	2005	10	2,28р.	2,00р.	0,29р.	336,90р.
26	12	2005	11	2,28р.	1,99р.	0,29р.	336,61р.
27	13	2005	12	2,28р.	1,99р.	0,29р.	336,32р.
28	***2005 Итог***			***27,42р.***	***24,01р.***	***3,41р.***	***336,32р.***
29	14	2006	1	2,28р.	1,99р.	0,30р.	336,02р.

Лист4 Лист1 Лист2 Лист3

Рис. 25.1. Это расписание амортизации займа наглядно знакомит с выплатами кредита под залог недвижимости, выданного сроком на 30 лет



Рис. 25.2. Первый этап работы приложения мастера расчета займа

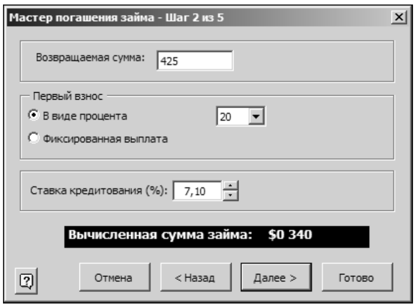


Рис. 25.3. Второй этап работы приложения мастера расчета займа

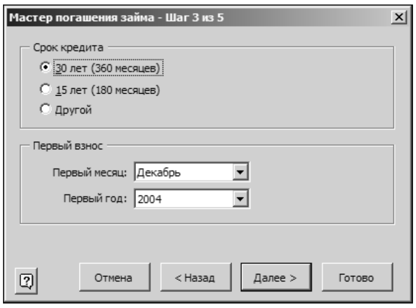


Рис. 25.4. Третий этап работы приложения мастера расчета займа

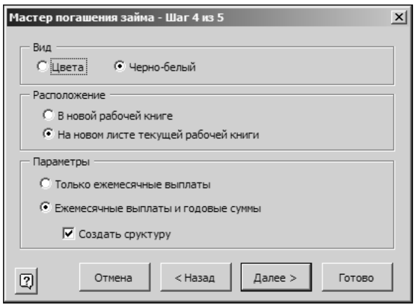


Рис. 25.5. Четвертый этап работы приложения мастера расчета займа

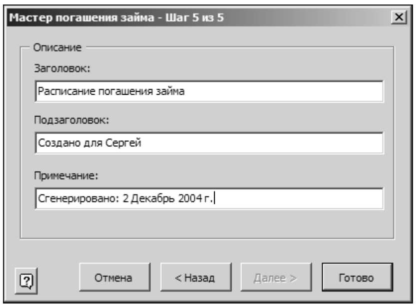


Рис. 25.6. Пятый этап работы приложения мастера расчета займа

Структура рабочей книги

Приложение мастера расчета займа состоит из следующих компонентов.

- `FormMain` — диалоговое окно `UserForm`, которое служит основным пользовательским интерфейсом.
- `FormHelp` — диалоговое окно `UserForm`, которое используется для отображения интерактивного справочного руководства.
- `HelpSheet` — рабочий лист, который содержит текст, отображаемый в интерактивном справочном руководстве.
- `ModMain` — модуль VBA, который содержит процедуру, отображающую основное диалоговое окно `UserForm`.

- ThisWorkbook — модуль кода этого объекта содержит процедуры обработки событий Open и BeforeClose объекта Workbook, что позволяет создавать и удалять необходимые меню.

Создание приложения мастера расчета займа

Приложение мастера расчета займа, начинавшееся с простой концепции, стало относительно сложный проектом. Основной его целью была демонстрация максимального количества методов разработки приложений, при этом необходимо было создать продукт, который приносит реальную пользу. Конечный результат можно было увидеть еще до начала разработки приложения, но основной идеей было создание конечного приложения, а не получение рассчитанных данных.

Главная цель приложения традиционно очень проста. Необходимо создать приложение, которое собирает от пользователя информацию и создает рабочий лист. Но в процессе разработки возникли идеи по расширению возможностей приложения. В итоге процесс несколько раз заходил в тупик. Кто-то может назвать эти эксперименты напрасной тратой времени, но они являются неотъемлемой частью обучения способам разработки приложений.

Проект был завершен в течение одного (длинного) дня. Еще несколько часов потрачено на тонкую настройку и тестирование приложения.

Как это работает

Приложение мастера расчета займа является надстройкой Excel, поэтому его необходимо устанавливать с помощью команды Сервис⇨Надстройки. Но приложение также будет работать, если его открыть с помощью команды Файл⇨Открыть.

ДОБАВЛЕНИЕ ОПЦИИ МЕНЮ

Когда рабочая книга открывается, процедура Workbook_Open добавляет новую опцию Мастер погашения займа в меню Сервис. Щелчок на этой опции меню приводит к запуску процедуры StartAmortizationWizard, которая отображает диалоговое окно FormMain.

Глава 23 содержит более подробную информацию о создании новых опций меню.

ИНИЦИАЛИЗАЦИЯ ДИАЛОГОВОГО ОКНА FORMMAIN

Процедура UserForm_Initialize для диалогового окна FormMain выполняет достаточно большой объем работы.

- Значение свойства Value элемента управления MultiPage устанавливается равным 0. Это обеспечивает отображение первой страницы элемента управления MultiPage, независимо от того, когда рабочая книга сохранялась в последний раз.
- Добавляются опции к раскрывающимся спискам трех элементов управления ComboBox, которые находятся в диалоговом окне.
- Вызывается процедура GetDefaults для установки значений, которые использовались в последний раз и хранятся в системном реестре (информация об этом приведена в разделе “Сохранение и получение значений по умолчанию” далее в этой главе).

- Проверяется активность рабочей книги. Если книга неактивна, то отключается элемент управления OptionButton, который позволяет пользователю создать новый рабочий лист в активной рабочей книге.
- Если рабочая книга активна, то в результате дополнительной проверки определяется защищенность структуры рабочей книги. Если это так, то процедура отключает элемент управления OptionButton, который позволяет создавать новый рабочий лист в активной рабочей книге.

ОБРАБОТКА СОБЫТИЙ В ПРОЦЕССЕ ОТОБРАЖЕНИЯ ДИАЛОГОВОГО ОКНА USERFORM

Модуль кода для диалогового окна FormMain содержит несколько процедур обработки событий Click и Change для элементов управления, которые находятся в диалоговом окне UserForm.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Щелчок на кнопках Назад и Далее определяет, какая из страниц элемента управления MultiPage будет отображена следующей. Процедура MultiPage_Change изменяет заголовок диалогового окна UserForm и включает или отключает кнопки Назад и Далее, в зависимости от того, какая страница элемента управления отображается в данный момент. Глава 15 содержит дополнительную информацию о методах программирования мастеров.

ОТОБРАЖЕНИЕ СПРАВОЧНОЙ ИНФОРМАЦИИ

Для отображения справочной информации существует несколько способов. В данном случае используется простая техника, в которой применяется диалоговое окно UserForm. Это диалоговое окно отображает текст, который хранится на рабочем листе. Можно заметить, что предоставляемая справочная информация зависит от контекста. Когда пользователь щелкает на кнопке Справка, отображается раздел справочного руководства, который относится к активной странице элемента управления MultiPage.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Дополнительная информация о методах использования рабочих листов для хранения справочной информации и запуска справочного руководства с помощью диалогового окна UserForm изложена в главе 24.

СОЗДАНИЕ НОВОГО РАБОЧЕГО ЛИСТА

Щелчок на кнопке Готово приведет к выполнению основной задачи приложения. Процедура обработки события Click для этой кнопки выполняет следующие действия.

- Вызывается функция DataIsValid, которая проверяет введенные пользователем данные на правильность. Если все введенные данные корректны, то функция возвращает значение True, и процедура продолжает свое выполнение. Если обнаружены некорректные входные данные, функция DataIsValid активизирует элемент управления, который содержит неверные данные, и создает окно сообщения с описанием проблемы (рис. 25.7).

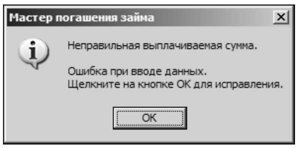


Рис. 25.7. Если пользователь ввел некорректные данные, то активизируется элемент управления, содержащий ошибку

- Если введенные пользователем данные корректны, то процедура создает новый рабочий лист в активной рабочей книге или в новой рабочей книге, что определяется выбором пользователя.
- Параметры кредитования (возвращаемая сумма, первый взнос, сумма займа, срок погашения и процентная ставка) записываются на этом рабочем листе. Здесь не обойтись без операторов `If`, так как сумма первого взноса может указываться и в виде процента от возвращаемой суммы, и в виде фиксированного значения.
- На рабочий лист заносятся заголовки столбцов.
- Первая строка формул вводится под строкой заголовков. Первая строка отличается от остальных, так как формулы в ней ссылаются на данные, которые хранятся в области данных о кредите. Остальные формулы ссылаются на значения в предыдущей строке. Обратите внимание, что в формулах используются именованные диапазоны. Эти имена определены на уровне листа, поэтому пользователь может хранить более одного расписания амортизации в одной и той же рабочей книге.
- Чтобы задать неименованные ссылки, используется формат R1C1 (что намного проще, чем определение фактического адреса ячейки).
- Вторая строка формул вводится на рабочий лист и копируется в нижние строки для каждого месяца.
- Если пользователь затребовал отображение ежегодных, а не ежемесячных данных, то в процедуре запускается метод `Subtotal` для расчета необходимых значений. Этот подход демонстрирует гибкость встроенных средств Excel, которые помогают избежать написания дополнительного кода.
- По причине того, что создание сумм в столбце Баланс теперь не имеет смысла, процедура заменяет формулу в столбце Баланс на формулу, которая возвращает значение баланса на ежегодной основе.
- Когда Excel подсчитывает суммы, она выделяет их с помощью границ. Если пользователь не требовал выделять суммы границами, то процедура использует метод `ClearOutline` для их отмены.
- Поле этого процедура производит форматирование информации, которая отображается в ячейках: в данном случае изменяется формат отображения чисел, а также применяется средство Автоформат, если пользователь требует выделять ячейки цветом.
- Далее процедура меняет ширину столбцов, блокирует заголовки и защищает формулы, а также несколько ключевых ячеек, значения в которых нельзя редактировать. После этого рабочий лист считается защищенным, хотя пароль не используется.
- Наконец, процедура `SaveDefaults` сохраняет сведения о состоянии элементов управления диалогового окна `UserForm` в системном реестре. Эти значения будут использоваться по умолчанию в следующий раз, когда пользователь примется за создание расписания амортизации займа (дополнительная информация приведена в разделе "Сохранение и получение значений по умолчанию" далее в этой главе).

Сохранение и получение значений по умолчанию

Если запустить это приложение, то можно заметить, что диалоговое окно `FormMain` всегда отображает значения, которые использовались в последний раз. Другими словами, диалоговое окно "запоминает" последние данные и применяет их в качестве новых значений по умолчанию. Таким образом упрощается создание не-

скольких сценариев “что-если” для амортизации кредита, которые отличаются только одним параметром. Подобный эффект достигается благодаря хранению значений в системном реестре и получению этих данных на этапе инициализации диалогового окна `UserForm`. Когда приложение запускается в первый раз, системный реестр не содержит значений, поэтому используются значения, принятые по умолчанию в элементах управления диалогового окна `UserForm`.

Следующая процедура, которая называется `GetDefaults`, циклически просматривает элементы управления в диалоговом окне `UserForm`. Если элемент управления имеет тип `TextBox`, `ComboBox`, `OptionButton`, `CheckBox` или `SpinButton`, то процедура вызывает функцию VBA `GetSetting` и считывает значения из системного реестра. Обратите внимание, что третий аргумент функции `GetSetting` представляет значение, которое используется в том случае, если в системном реестре не найдено требуемых параметров. В результате применяется значение, которое указывается для элемента управления на этапе проектирования. `APPNAME` является глобальной константой, содержащей название приложения.

```
Sub GetDefaults()
    Dim ctl As Control
    Dim CtrlType As String

    For Each ctl In Me.Controls
        CtrlType = TypeName(ctl)
        If CtrlType = "TextBox" Or _
           CtrlType = "ComboBox" Or _
           CtrlType = "OptionButton" Or _
           CtrlType = "CheckBox" Or _
           CtrlType = "SpinButton" Then
           ctl.Value = VBA.GetSetting _
             (APPNAME, "Defaults", ctl.Name, ctl.Value)
        End If
    Next ctl
End Sub
```

На рис. 25.8 показано, как эти значения выглядят при запуске системного реестра.

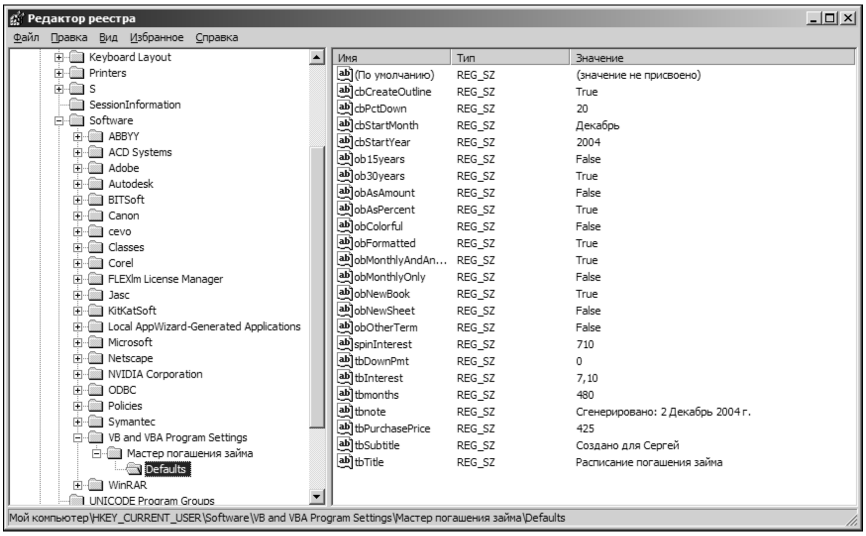


Рис. 25.8. Системный реестр хранит значения по умолчанию элементов управления мастера

Представленная далее процедура — `SaveDefaults` — подобна предыдущей. Она использует оператор VBA `SaveSetting` для записи текущих значений в системный реестр.

```
Sub SaveDefaults()
    Dim ctl As Control
    Dim CtrlType As String

    For Each ctl In Me.Controls
        CtrlType = TypeName(ctl)
        If CtrlType = "TextBox" Or _
           CtrlType = "ComboBox" Or _
           CtrlType = "OptionButton" Or _
           CtrlType = "CheckBox" Or _
           CtrlType = "SpinButton" Then
            SaveSetting APPNAME, "Defaults", ctl.Name, ctl.Value
        End If
    Next ctl
End Sub
```

Обратите внимание, что в коде используется функция `CStr`, которая преобразует каждую настройку в строку. Таким образом, самым предотвращаются неполадки у тех пользователей, которые установили другие региональные стандарты.

И функция `GetSetting`, и оператор `SaveSetting` всегда используют приведенный ниже раздел системного реестра для хранения значений:

```
HKEY_CURRENT_USER\Software\VB and VBA Program Settings\
```

Потенциальные улучшения

Считается, что создание приложения никогда не завершается — просто прекращается работа над внесением улучшений. Даже затратив немного времени, можно придумать несколько улучшений, которые можно внести в приложение мастера расчета займа.

- Добавить элемент управления, который позволит пользователю ознакомиться с текущим состоянием погашения процента и общей суммы кредита.
- Добавить возможность использования дробной процентной ставки, что позволит создавать долгосрочные прогнозы, рассчитанные для разных значений процентной ставки.
- Добавить больше параметров форматирования (например, отключить десятичные разряды и отобразить символы доллара).
- Предоставить пользователю возможность указывать текст верхнего и нижнего колонтитула страницы.

Концепции разработки приложений

Зачастую сложно понять логику приложения, которое разработано кем-то другим. Для того чтобы помочь разобраться с основами создания рассмотренного приложения, в исходный код приложения обычно добавляются комментарии. Но если вам важно понять принципы, лежащие в основе определенного приложения, то воспользуйтесь отладчиком для пошагового выполнения исходного кода.

Приложение мастера расчета займа в полной мере демонстрирует основные методы и концепции разработки приложений в Excel.

- Создание собственных опций меню, которые отображаются только при открытии определенной рабочей книги или надстройки.

- Использование диалоговых окон `UserForm`, предназначенных для получения информации от пользователя. В отношении принципов своей работы они подобны встроенным мастерам Excel.
- Динамическая установка свойства `Enabled` элементов управления.
- Связывание элемента управления `TextBox` с элементом управления `SpinButton`.
- Отображение интерактивного справочного руководства.
- Именование ячеек с помощью кода VBA.
- Создание и копирование формул с помощью кода VBA.
- Чтение и запись данных в системный реестр.

Заключение

Разработка приложений Excel, ориентированных на пользователя, является достаточно сложной задачей. Необходимо точно знать, как конечные пользователи используют (и приводят в неработоспособное состояние) приложения в реальной жизни. Хотя мы и пытались создать приложение, полностью защищенное от неквалифицированных пользователей, оно не подвергалось активному тестированию в реальных условиях, поэтому при определенных условиях данное приложение может неожиданно завершить свое выполнение.

Список действий, которые необходимо выполнить при разработке приложения

При разработке приложения, ориентированного на пользователя, следует помнить о некоторых основных моментах. Следующий список действий будет служить вам полезным напоминанием.

- *Производит ли приложение восстановление данных после завершения своей работы?* Удостоверьтесь, что приложение по завершении своей работы восстанавливает состояние панелей инструментов и меню.
- *Поддерживают ли диалоговые окна работу с помощью клавиатуры?* Не забудьте добавить комбинации клавиш, а также проверьте правильность порядка перехода между элементами управления.
- *Делает ли приложение предположения о существовании папок?* Если приложение читает или записывает информацию в файлы, то следует ожидать, что папка по умолчанию уже существует.
- *Делается ли предположение о том, что больше не открыт ни один рабочий лист?* Если приложение является единственной рабочей книгой, которая открыта во время тестирования, то может оказаться, что не рассмотрены случаи, когда открыто несколько рабочих книг.
- *Осуществлялись ли предположение о видимости рабочей книги?* Excel можно использовать и тогда, когда на экране не отображена ни одна рабочая книга.
- *Делались ли попытки оптимизации приложения?* Например, можно увеличить быстродействие приложения, если явно указать тип переменных.
- *Является ли адекватной справочная документация для созданной процедуры?* Существует ли возможность понять код, к которому необходимо будет вернуться через шесть месяцев после его создания?
- *Предоставляется ли адекватная документация для конечного пользователя?* Выполнение этого условия сокращает количество вопросов со стороны конечных пользователей.
- *Выделено ли время для проверки работы приложения?* Приложение не может быть идеальным сразу же после разработки. Поэтому необходимо выделить время для его улучшения.

Часть VII

Другие темы

В этой части...

Глава 26

Вопросы совместимости

В ЭТОЙ ГЛАВЕ...

Следующие вопросы являются основной темой настоящей главы.

- Запуск приложений Excel 2003 в более ранних версиях программы
- Сложности, возникающие при разработке приложения Excel для интернационального использования

Если разрабатываемые вами приложения используют средства Excel 2003 и предназначены для применения с такой же версией Excel, то эту главу вам можно не читать. Но если приложение должно выполняться под управлением более старых версий Excel, в Excel для Macintosh, а также в интернациональных версиях Excel, то рекомендуем ознакомиться с вопросами несовместимости данных.

Что такое совместимость

Совместимость — это часто используемый термин в среде разработчиков компьютерных приложений. Данный термин обозначает правильность выполнения приложения в различных условиях. Эти условия определяются настройками аппаратного или программного обеспечения, а также принципами их взаимодействия. Например, программное обеспечение, которое создано специально для 32-битовой системы (например, Windows XP), не будет работать под управлением 16-битовой более старой версии Windows (3.x). Другими словами, 32-битовые приложения не совместимы с Windows 3.x. Как несложно догадаться, программное обеспечение, созданное для Windows, не будет автоматически запускаться под управлением таких операционных систем, как MacOS или Linux.

В этой главе вопросы совместимости рассматриваются более детально; под этим подразумевается выполнение приложений Excel 2003 в более ранних версиях Excel для Windows и Excel для Macintosh. Тот факт, что две версии Excel используют одинаковый формат файлов, не гарантирует, что совместимость содержимого этих файлов будет полной. Например, Excel 97, Excel 2000 и Excel 2002, Excel 2003, а также Excel 98 для Macintosh поддерживают одинаковый формат файлов, но при этом конфликты совместимости возникают довольно часто. Тот факт, что определенная версия Excel может открыть файл рабочей книги надстройки, не означает, что она будет обрабатывать инструкции VBA, которые сохранены в этом файле.

Вопросы совместимости более важны, чем кажется на первый взгляд. С проблемами совместимости можно столкнуться даже в пределах одной версии Excel. Например, Excel 2000 существует в виде как минимум трех подверсий: первоначальный выпуск, версия SR-1 и еще одна специальная версия, которая называется SR-1a. Дополнительные версии предназначены для исправления ошибок исходной программы — они вносят малозаметные визуальные изменения, хотя часто кардинально изменяют поведение Excel. Таким образом, невозможно гарантировать, что приложение, разработанное в базовой версии, будет безошибочно работать в Excel 2000 подверсии SR-1.

Итак, Excel постоянно изменяется, совершенствуется и обновляется, поэтому не существует способа обеспечить полную совместимость создаваемых приложений. К сожалению, автоматически обеспечить совместимость приложения с различными версиями невозможно. В большинстве случаев необходимо выполнить достаточно большой объем дополнительной работы, чтобы достигнуть приемлемого уровня совместимости.

Проблемы совместимости

Вам необходимо помнить о пяти категориях проблем совместимости. Эти категории перечислены ниже и детально рассматриваются далее в этой главе.

- *Несовместимость форматов файлов.* Рабочие книги могут сохраняться в нескольких форматах Excel. Порой невозможно открыть рабочую книгу, которая сохранена в более поздней версии Excel.
- *Несовместимость новых средств.* Очевидно, что новые возможности, которые появились в поздних версиях Excel, не могут использоваться в старых версиях программы.
- *32-битовые и 16-битовые архитектуры.* Если вами используются функции Windows API и приложение должно работать с 16-битовой версией Excel (Excel 5), то необходимо обратить внимание и на эту проблему.
- *Совместимость Windows и Macintosh.* Если приложение должно работать под управлением обеих платформ, то придется потратить намного больше времени для решения всех проблем совместимости.
- *Вопросы языковой совместимости.* Если приложение будет использоваться в иноязычных версиях Excel, то обратите внимание на целый ряд дополнительных вопросов языковой совместимости.

После изучения этой главы вам должно стать ясно, что существует один гарантированный способ достижения совместимости: приложение необходимо протестировать на всех целевых платформах, а также во всех интересующих версиях Excel. Зачастую это просто невозможно. Однако существуют некоторые способы, которые помогают разработчику частично гарантировать работу приложения в различных версиях Excel.

НА ЗАМЕТКУ

Если вы ожидаете, что в этой главе будет приведен список всех проблем несовместимости версий Excel, то не вводите себя в заблуждение. Такого списка просто не существует, и его создание является практически невозможным.

СОВЕТ

Хорошим источником информации о потенциальных проблемах совместимости можно считать интерактивную базу знаний Microsoft, которая расположена по адресу `http://search.support.microsoft.com`. Эта база поможет идентифицировать проблему, которая возникает в определенной версии Excel.

Поддерживаемые форматы файлов Excel

Программа Excel позволяет сохранять рабочие книги в формате, который поддерживается более ранними версиями. Кроме того, можно сохранить рабочую книгу в формате “двух версий” (в одном файле поддерживается два формата). Использование такого подхода приводит к увеличению размера сохраняемого файла.

Если приложение должно работать с более ранними версиями Excel, то необходимо сохранить файл рабочей книги в соответствующем формате. Excel 2003 поддерживает следующие форматы файлов.

- *Книга Microsoft Excel (*.xls)*. Стандартный формат для файлов Excel 2003. Он поддерживается в Excel 97, Excel 2000–2003.
- *Книга Microsoft Excel 5.0/95*. Формат, который поддерживается в Excel 5 и в более поздних версиях.
- *Книга Microsoft Excel 97-2000 & 5.0/95*. Двойной формат, который поддерживается в Excel 5 и в более поздних версиях.
- *Файл Microsoft Excel 4.0 (*.xls)*. Может открываться в Excel 4 и более поздних версиях. Используется для сохранения одного рабочего листа.
- *Файл Microsoft Excel 3.0 (*.xls)*. Может открываться в Excel 3 и более поздних версиях. Используется для сохранения одного рабочего листа.
- *Файл Microsoft Excel 2.1 (*.xls)*. Может открываться в Excel 2.1 и в более поздних версиях. Используется для сохранения одного рабочего листа.
- *Книга Microsoft Excel 4 (*.xlw)*. Может открываться в Excel 4 и более поздних версиях. Это формат используется для сохранения нескольких рабочих листов, но они отличаются от рабочей книги Excel 5.

Код VBA также применяется для получения доступа к свойству `FileFormat` объекта `Workbook`, определяющего формат файла определенной рабочей книги. Например, приведенный ниже оператор отображает значение, которое представляет формат файла активной рабочей книги.

```
MsgBox ActiveWorkbook.FileFormat
```

Для указания значения свойства `FileFormat` используются предопределенные константы. Например, оператор, который приводится ниже, отображает значение `True`, если активная рабочая книга имеет формат Excel 5.

```
MsgBox ActiveWorkbook.FileFormat = xlExcel5
```

В табл. 26.1 представлен список констант и их значений, предназначенных для представления в VBA различных форматов файлов Excel.

Таблица 26.1. Константы и их значения для различных форматов файлов Excel

Версии Excel	Константы	Значения
Excel 2.1	`xlExcel2`	16
Excel 3	`xlExcel3`	29
Excel 4	`xlExcel4Workbook`	35
Excel 5	`xlExcel5`	39
Excel 95/97	`xlExcel9795`	43
Excel в формате HTML	`xlHtml`	44
Надстройка Excel	`xlAddIn`	18
Excel 97/2000/2002/2003	`xlWorkbookNormal`	-4143

Избегайте использования новых возможностей

Если приложение должно работать в нескольких версиях Excel (Excel 2003 и более ранних), то следует избегать использования новых возможностей, добавленных после выхода самой ранней версии Excel, которую необходимо поддерживать. Еще одной альтернативой является выборочное использование новых возможностей. Другими словами, приложение должно определить, какая версия Excel применяется, и на основании этого принимать решение об использовании нового средства.

Разрабатывая приложения с помощью VBA, не следует использовать объекты, методы и свойства, которые не доступны в более ранних версиях. Самым безопасным подходом является разработка приложения на основе "наибольшего общего знаменателя". Чтобы обеспечить совместимость с Excel 2002, 2000, Excel 97 и Excel 95, необходимо для разработки приложения использовать Excel 95, после чего продукт должен быть всесторонне протестирован в остальных средах.

Если приложение будет использоваться в Excel 95, то диалоговые окна `UserForm` в него добавлять нельзя. Пользовательские диалоговые окна появились только в Excel 97. Вместо них вам придется работать с диалоговыми листами.

Определение версии Excel

Свойство `Version` объекта `Application` определяет номер версии Excel. Возвращаемое значение является строкой, поэтому часто требуется преобразовать его в число. Функция VBA `Val` как нельзя лучше подходит для выполнения этой задачи. Следующая функция возвращает значение `True`, если пользователь запускает приложение в Excel 2002 или более поздней версии (Excel 2002 соответствует версии 10).

```
Function XL10OrLater()
    XL10OrLater = Val(Application.Version) >= 10
End Function
```

Поддержка платформы Mac

Чаще всего проблемы совместимости касаются компьютеров Macintosh. По причине того, что Excel для Macintosh составляет небольшую часть рынка Excel, многие разработчики просто игнорируют эту платформу. Microsoft обеспечила совместимость файлов Excel между двумя платформами. Однако в разных платформах поддерживается неодинаковый набор возможностей, а совместимость макросов VBA далека от идеальной.

Можно создать код VBA, который будет определять, на какой платформе работает приложение. Следующая функция получает доступ к свойству `OperatingSystem` объекта `Application` и возвращает значение `True`, если используется одна из версий Windows (т.е. если возвращаемая строка содержит значение `"Win"`).

```
Function WindowsOS() As Boolean
    If Application.OperatingSystem like "*Win*" Then
        WindowsOS = True
    Else
        WindowsOS = False
    End If
End Function
```

Между Windows-версией и Mac-версией Excel существует ряд отличий. Некоторые из них можно считать “косметическими” (например, по умолчанию используются разные системные шрифты). Но существуют и более серьезные проблемы. Например, в состав Excel для Macintosh не входят компоненты ActiveX. Кроме того, Excel для Macintosh использует систему дат “1904”, поэтому рабочие книги, в которых применяется система дат по умолчанию, могут опережать время на 4 года. Excel для Windows по умолчанию использует систему дат “1900”. Это означает, что в Macintosh дата 1 будет означать 1 января 1904 года. В то же время в Windows такая дата будет означать 1 января 1900 года.

Еще одно ограничение касается вызова функций Windows API. В Excel для Macintosh они просто не работают. Если приложение полностью зависит от таких функций, то вам придется разрабатывать дополнительные способы выполнения заданий, возложенных на приложение.

Если код работает с именами файлов, значит, необходимо указать путь с использованием подходящего разделителя (двоеточие в Macintosh и обратная косая черта для Windows). Наиболее правильно динамически определить подходящий разделитель в пути с помощью кода VBA. Представленный ниже оператор назначает символ разделителя пути переменной `PathSep`.

```
PathSep = Application.PathSeparator
```

После выполнения этого оператора код может использовать переменную `PathSep` для замещения строго определенного разделителя.

Вместо того, чтобы создавать один файл, совместимый с обеими платформами, многие разработчики принимают решение о сохранении двух версий приложения для каждой из платформ. Сначала создается приложение для одной платформы (обычно это Excel для Windows), после чего оно модифицируется для работы на другой платформе. Другими словами, вам придется отлаживать две различные версии приложения.

Существует только один способ обеспечения совместимости приложения с Excel для Macintosh: приложение должно быть всесторонне протестировано на этой платформе. Будьте готовы к разработке обходных путей, которые заменят отсутствующие или неработающие возможности.

Создание интернациональных приложений

Последний вопрос совместимости касается языковых и интернациональных настроек. Excel поставляется в различных языковых версиях. Следующий оператор выводит значение кода страны текущей версии Excel.

```
MsgBox Application.International(xlCountryCode)
```

Версия United States/English имеет код 1. Остальные коды стран перечислены в табл. 26.2.

Если вы знаете, на каком языке общаются люди, которые будут использовать приложение, удостоверьтесь, что в диалоговых окнах используются сообщения на соответствующем языке. Кроме того, необходимо определить правильные разделители десятичных дробей и тысячных разрядов, применяемых в стране пользователя. В Соединенных Штатах для этого практически всегда используются точка и запятая. Но пользователи других стран могут обращаться к другим разделителям. Помните также о различиях в форматах представления даты и времени. Соединенные Штаты входят в число стран, которые используют формат даты `месяц/число/год`.

Таблица 26.2. Коды языков, поддерживаемых в Excel

Язык	Код страны
Английский	1
Русский	7
Греческий	30
Голландский	31
Французский	33
Испанский	34
Венгерский	36
Итальянский	39
Чешский	42
Датский	45
Шведский	46
Норвежский	47
Польский	48
Немецкий	49
Португальский (Бразилия)	55
Тайский	66
Японский	81
Корейский	82
Вьетнамский	84
Упрощенный китайский	86
Турецкий	90
Индийский	91
Урду	92
Португальский	351
Финский	358
Традиционный китайский	886
Арабский	966
Иврит	972
Фарси	982

Если разрабатывается приложение, которое будет применяться только пользователями одной компании, то о вопросах интернациональной совместимости можно не задумываться. Но если компания имеет офисы по всему миру или планируется распространение приложения за пределы страны, то стоит обратить внимание на ряд вопросов, что позволит обеспечить правильную работу приложения. Эти вопросы рассматриваются в данном разделе.

Многоязыковые приложения

Весьма очевидным моментом является язык общения, который будет использоваться в приложении. Например, если приложение отображает одно или несколько диалоговых окон, то может возникнуть необходимость в отображении текста на языке пользователя. К счастью, это несложно (если, конечно, вы владеете соответствующим языком).

На прилагаемом к книге компакт-диске содержится мастер диалоговых окон, который поддерживает три языка: английский, испанский и немецкий. Этот мастер основан на примере диалогового окна, который рассматривался в главе 14.

Первый этап работы мастера заключается в отображении элемента управления `OptionButton`, позволяющего пользователю указать язык общения. Текст на трех языках хранится на рабочем листе.

На рис. 26.1–26.3 показаны диалоговые окна `UserForm`, которые отображают текст на трех языках.

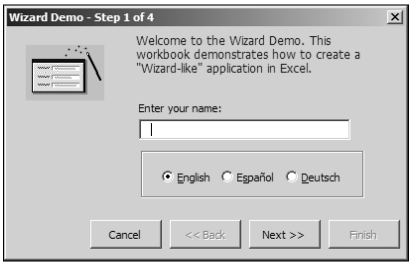


Рис. 26.1. Демонстрация работы мастера на английском языке

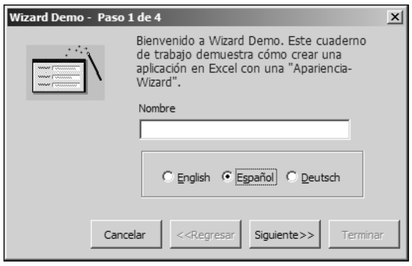


Рис. 26.2. Демонстрация работы мастера на испанском языке

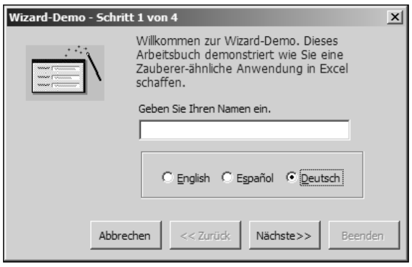


Рис. 26.3. Демонстрация работы мастера на немецком языке

Язык в VBA

Как правило, задумываться о языке при создании кода VBA не стоит. Excel использует две библиотеки объектов: Excel Object library и VBA Object library. При установке Excel регистрируется соответствующая версия этих библиотек. Данные версии применяются по умолчанию (они не зависят от языковой версии Excel).

Использование “локальных” свойств

Если код будет отображать информацию рабочего листа, например, диапазон адресов, то, скорее всего, необходимо использовать язык общения с пользователем. Например, следующий оператор отображает адрес выделенного диапазона.

```
MsgBox Selection.Address
```

Для интернациональных приложений наиболее удачным подходом будет использование свойства `AddressLocal` вместо свойства `Address`.

```
MsgBox Selection.AddressLocal
```

Рекомендуется обеспечить соответствие стилей ссылок и стиля, выбранного пользователем (A1 или R1C1). Этого можно достичь с помощью следующего оператора.

```
MsgBox Selection.AddressLocal _
    (ReferenceStyle:=Application.ReferenceStyle)
```

"Локальные" версии свойств предоставляют еще несколько возможностей. Эти свойства показаны в табл. 26.3 (дополнительную информацию можно получить, обратившись к справочному руководству).

Таблица 26.3. "Локальные" версии свойств

Свойство	"Локальная" версия	Что возвращает
Address	AddressLocal	Адрес
Category	CategoryLocal	Категорию функции
Formula	FormulaLocal	Формулу
Name	NameLocal	Имя
NumberFormat	NumberFormatLocal	Формат числа
RefersTo	RefersToLocal	Ссылку

Идентификация настроек системы

Очень редко случается так, что система конечного пользователя настроена так же, как и система, в которой происходила разработка приложения. Для интернациональных приложений необходимо получить информацию о следующих параметрах настройки системы.

- Разделитель десятичных знаков — символ, который используется для отделения десятичной части значения.
- Разделитель тысяч — символ, который используется для разделения триад цифр.
- Разделитель списка — символ, который используется для разделения элементов списка.

Текущее значение параметра настройки можно получить с помощью свойства International объекта Application. Например, следующий оператор отображает разделитель десятичных знаков, который не всегда является запятой.

```
MsgBox Application.International(xlDecimalSeparator)
```

Существует 45 интернациональных параметров настройки, доступ к которым можно получить с помощью свойства International. Эти параметры перечислены в табл. 26.4.

Таблица 26.4. Константы свойства International

Константа	Что определяет
xlCountryCode	Языковую версию Microsoft Excel
xlCountrySetting	Текущие установки страны в папке Панель управления
xlDecimalSeparator	Разделитель десятичной части
xlThousandsSeparator	Разделитель нулей или триад цифр
xlListSeparator	Разделитель списка
xlUpperCaseRowLetter	Символ сроки в верхнем регистре (для ссылок в стиле R1C1)
xlUpperCaseColumnLetter	Символ столбца в верхнем регистре

Константа	Что определяет
`xlLowerCaseRowLetter`	Символ строки в нижнем регистре
`xlLowerCaseColumnLetter`	Символ столбца в нижнем регистре
`xlLeftBracket`	Символ, который используется вместо открывающей квадратной скобки ([) в относительных ссылках стиля `R1C1`
`xlRightBracket`	Символ, который используется вместо закрывающей квадратной скобки (]) в относительных ссылках стиля R1C1
`xlLeftBrace`	Символ, который используется вместо открывающей фигурной скобки ({) в массивах символов
`xlRightBrace`	Символ, который используется вместо закрывающей фигурной скобки (}) в массивах символов
`xlColumnSeparator`	Символ, который используется для разделения столбцов в массивах символов
`xlRowSeparator`	Символ, который используется для разделения строк в массивах символов
`xlAlternateArraySeparator`	Символ, который используется в качестве альтернативного разделителя массивов, если текущий разделитель применен в качестве разделителя десятичной части
`xlDateSeparator`	Разделитель даты (/)
`xlTimeSeparator`	Разделитель времени (:)
`xlYearCode`	Символ года в формате числа (y)
`xlMonthCode`	Символ месяца (m)
`xlDayCode`	Символ числа (d)
`xlHourCode`	Символ часа (h)
`xlMinuteCode`	Символ минуты (m)
`xlSecondCode`	Символ секунды (s)
`xlCurrencyCode`	Символ валюты
`xlGeneralFormatName`	Название основного формата числа
`xlCurrencyDigits`	Количество десятичных цифр, которые отображаются в формате валюты
`xlCurrencyNegative`	Значение, которое отображает формат валюты для отрицательных значений
`xlNoncurrencyDigits`	Число десятичных цифр, которое используется в обычных числах
`xlMonthNameChars`	Всегда возвращает три символа для обеспечения обратной совместимости. Аббревиатуры месяцев предоставляются Windows и могут быть любой длины
`xlWeekdayNameChars`	Всегда возвращает три символа для обеспечения обратной совместимости. Аббревиатуры дней недели предоставляются Windows и могут быль любой длины
`xlDateOrder`	Целое число, которое отображает порядок элементов в формате даты
`xl24HourClock`	`True`, если в системе используется 24-часовая система. `False`, если в системе используется 12-часовая система
`xlNonEnglishFunctions`	`True`, если система не отображает функции на английском языке

Окончание табл. 26.4

Константа	Что определяет
`xlMetric`	`True`, если используется метрическая система. `False`, если используется британская система измерения
`xlCurrencySpaceBefore`	`True`, если перед символом валюты добавляется пробел
`xlCurrencyBefore`	`True`, если символ валюты указывается до значения. `False` — в противном случае
`xlCurrencyMinusSign`	`True`, если для представления отрицательного денежного значения используется знак минус. `False`, если для этого используются скобки
`xlCurrencyTrailingZeros`	`True`, если в нулевых денежных значениях используются нули, добавленные в конце
`xlCurrencyLeadingZeros`	`True`, если в нулевых денежных значениях используются нули, добавленные вначале
`xlMonthLeadingZero`	`True`, если при выводе месяца отображается дополняющий ноль
`xlDayLeadingZero`	`True`, если при выводе дня отображается дополняющий ноль
`xl4DigitYears`	`True`, если в системе используются четырехзначные годы, `False` — в противном случае (двузначные годы)
`xlMDY`	`True`, если для отображения даты используется формат месяц-день-год, `False`, если используется формат день-месяц-год
`xlTimeLeadingZero`	`True`, если при отображении времени используются дополняющие нули

Параметры настройки даты и времени

Если приложение отображает отформатированные даты, которые будут использоваться в других странах, то необходимо удостовериться, что формат даты знаком пользователю. Наиболее удачным подходом считается указание даты с помощью функции VBA `DateSerial`, что перекладывает бремя форматирования данных “на плечи” Excel (при этом используется краткий формат даты).

Следующая процедура использует функцию `DateSerial` для присвоения даты переменной `StartDate`. Дата заносится в ячейку `A1` в локальном кратком формате.

```
Sub WriteDate()
    Dim StartDate As Date
    StartDate = DateSerial(2001, 2, 15)
    Range("A1") = StartDate
End Sub
```

Если необходимо дополнительно изменить формат даты, то можно создать код, который будет выполнять эту задачу после записи даты в ячейку. Excel предоставляет несколько именованных форматов даты и времени, а также несколько именованных форматов представления чисел. Все они рассматриваются в справочном руководстве.

Глава 27

Управление файлами с помощью VBA

В ЭТОЙ ГЛАВЕ...

В настоящей главе описываются средства VBA, предназначенные для выполнения распространенных операций над любыми файлами, а также для непосредственного управления текстовыми файлами. Обзор средств VBA по управлению файлами приведен ниже.

- Выполнение привычных операций с файлами с помощью традиционных методов, а также объекта `FileSearch`.
- Способы открытия файлов.
- Примеры записи и чтения файлов с использованием VBA.
- Код, предназначенный для импортирования более 256 столбцов данных в рабочую книгу.
- Пример кода, предназначенного для экспорта диапазона в формат HTML.

Некоторые приложения Excel работают с несколькими файлами. Например, у вас может возникнуть необходимость в получении списка файлов, которые находятся в определенной папке; также может понадобится удалить или переименовать файлы и т.д. Конечно, Excel импортирует и экспортирует файлы в нескольких текстовых форматах, но часто требуется выполнить такие действия, с которыми встроенные инструменты Excel справиться не могут. Например, у вас возникает необходимость в импортировании более 256 столбцов данных. Или файл содержит другой разделитель данных, например, обратную косую черту.

Часто выполняемые операции

Excel предоставляет ряд возможностей по выполнению распространенных операций над файлами.

- Можно использовать “традиционные” операторы и функции VBA. Этот метод работает во всех версиях Excel.
- Можно применить объект `FileSearch`, с которым работать намного проще. Он предоставляет несколько дополнительных возможностей. Этот метод используется в Excel 97 и более поздних версиях программы.
- Можно обратиться к помощи объекта `FileSystemObject`, который задействует библиотеку Microsoft Scripting Library. Этот метод поддерживается в Excel 2000 и более поздних версиях программы.

В следующем разделе рассматриваются все три метода и приводятся примеры использования каждого из них.

Команды VBA по управлению файлами

Команды VBA, которые используются для управления файлами, перечислены в табл. 27.1.

Таблица 27.1. Команды VBA, предназначенные для управления файлами

Команда	Назначение
`ChDir`	Изменения текущей папки
`ChDrive`	Изменения текущего диска
`Dir`	Возвращает имя файла или папке, которое соответствует определенному шаблону или атрибуту файла
`FileCopy`	Копирует файл
`FileDateTime`	Возвращает дату и время последнего изменения файла
`FileLen`	Возвращает размер файла (в байтах)
`GetAttr`	Возвращает значение, определяющее атрибуты файла
`Kill`	Удаляет файл
`MkDir`	Создает новую папку
`Name`	Переименовывает файл или папку
`RmDir`	Удаляет пустую папку
`SetAttr`	Изменяет атрибуты файла

Далее будут приведены примеры, которые демонстрируют применение этих команд.

ОПРЕДЕЛЕНИЕ ФАКТА СУЩЕСТВОВАНИЯ ФАЙЛА

Представленная ниже функция возвращает значение `True`, если определенный файл существует. Если файл не существует, функция возвращает значение `False`. Когда функция `Dir` возвращает пустую строку, то файл невозможно найти. В этом случае функция возвращает значение `False`.

```
Function FileExists(fname) As Boolean
    If Dir(fname) <> "" Then _
      FileExists = True _
      Else FileExists = False
End Function
```

Аргумент функции `FileExists` состоит из полного пути и имени файла. Функция может использоваться на рабочем листе, а также вызываться из кода VBA.

ОПРЕДЕЛЕНИЕ ФАКТА СУЩЕСТВОВАНИЯ ПУТИ

Следующая функция возвращает значение `True`, если указанный путь существует. В противном случае функция возвращает значение `False`.

```
Function PathExists(pname) As Boolean
'   Возвращает TRUE, если путь существует
    On Error Resume Next
    PathExists = GetAttr(pname) And vbDirectory = vbDirectory
End Function
```

ОТОБРАЖЕНИЕ СПИСКА ФАЙЛОВ В ПАПКЕ

Следующая процедура отображает (на активном рабочем листе) список файлов, которые содержатся в определенной папке. Кроме того, отображается размер каждого файла, а также дата последнего изменения.

```
Sub ListFiles()
    Directory = "c:\windows\"
    r = 1

'   Вставка заголовков
    Cells(r, 1) = "Имя файла"
    Cells(r, 2) = "Размер"
    Cells(r, 3) = "Дата/Время"
    Range("A1:C1").Font.Bold = True

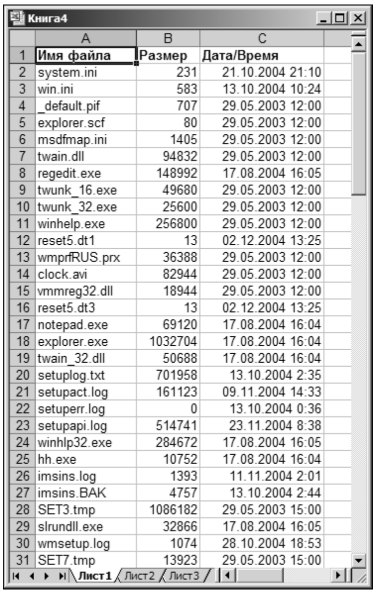
'   Получение первого файла
    f = Dir(Directory, 7)
    Do While f <> ""
        r = r + 1
        Cells(r, 1) = f
        Cells(r, 2) = FileLen(Directory & f)
        Cells(r, 3) = FileDateTime(Directory & f)
    '   Получение следующего файла
        f = Dir
    Loop
End Sub
```

На рис. 27.1 показан результат выполнения этой процедуры.

Обратите внимание, что процедура использует функцию `Dir` дважды. Первый раз — для получения имени первого файла, следующий — для получения остальных имен файлов. Если больше файлов не найдено, функция `Dir` возвращает пустую строку.

НА КОМПАКТ-ДИСКЕ

На прилагаемом к книге компакт-диске вы найдете более сложную версию этой процедуры. Она позволяет указывать папку в диалоговом окне.



Книга4

	A	B	C
1	Имя файла	Размер	Дата/Время
2	system.ini	231	21.10.2004 21:10
3	win.ini	583	13.10.2004 10:24
4	_default.pif	707	29.05.2003 12:00
5	explorer.scf	80	29.05.2003 12:00
6	msdfmap.ini	1405	29.05.2003 12:00
7	twain.dll	94832	29.05.2003 12:00
8	regedit.exe	148992	17.08.2004 16:05
9	twunk_16.exe	49680	29.05.2003 12:00
10	twunk_32.exe	25600	29.05.2003 12:00
11	winhelp.exe	256800	29.05.2003 12:00
12	reset5.dt1	13	02.12.2004 13:25
13	wmprfRUS.prx	36388	29.05.2003 12:00
14	clock.avi	82944	29.05.2003 12:00
15	vmmreg32.dll	18944	29.05.2003 12:00
16	reset5.dt3	13	02.12.2004 13:25
17	notepad.exe	69120	17.08.2004 16:04
18	explorer.exe	1032704	17.08.2004 16:04
19	twain_32.dll	50688	17.08.2004 16:04
20	setuplog.txt	701958	13.10.2004 2:35
21	setupact.log	161123	09.11.2004 14:33
22	setuperr.log	0	13.10.2004 0:36
23	setupapi.log	514741	23.11.2004 8:38
24	winhlp32.exe	284672	17.08.2004 16:05
25	hh.exe	10752	17.08.2004 16:04
26	imsins.log	1393	11.11.2004 2:01
27	imsins.BAK	4757	13.10.2004 2:44
28	SET3.tmp	1086182	29.05.2003 15:00
29	slrundll.exe	32866	17.08.2004 16:05
30	wmsetup.log	1074	28.10.2004 18:53
31	SET7.tmp	13923	29.05.2003 15:00

Лист1 Лист2 Лист3

Рис. 27.1. Результат выполнения процедуры `ListFiles`

Функция `Dir` в качестве первого аргумента принимает групповые символы. Чтобы получить список (например, файлов Excel), используйте следующий оператор.

```
f = Dir(Directory & "*.xl?", 7)
```

Данный оператор приводит к получению первого имени файла в текущей папке. Это имя соответствует шаблону `*.xl?`. Второй аргумент функции `Dir` позволяет задать атрибуты файлов. Аргумент, равный 7, приводит к получению имен файлов, которые не имеют атрибутов, файлов, предназначенных только для чтения, а также скрытых и системных файлов. Для получения дополнительной информации обратитесь к диалоговому справочному руководству.



Если вам необходимо отобразить список файлов, чтобы позволить пользователю выбрать один из них, то этот метод не самый удачный. Обратитесь к методу `GetOpenFileName`, который рассматривался в главе 12.

Использование объекта FileSearch

Объект `FileSearch` является членом библиотеки объектов Microsoft Office. Этот объект снабжает код VBA функциональностью диалогового окна поиска файлов. Например, можно использовать этот объект для поиска файлов, которые соответствуют указанному шаблону (например, `*.xls`), или даже для нахождения файлов, которые содержат определенный текст. Данный объект применяется в Excel 97 и в более поздних версиях программы.

В табл. 27.2 отображены основные методы и свойства, которые предоставляются объектом `FileSearch`. Дополнительная информация приведена в справочном руководстве.

Таблица 27.2. Свойства и методы объекта *FileSearch*

Свойство или метод	Назначение
`FileName`	Имя файла, который необходимо найти (допускается использование групповых символов)
`FoundFiles`	Возвращает объект, который содержит имена найденных файлов
`LookIn`	Папка, в которой производится поиск
`SearchSubFolders`	Имеет значение `True`, если необходимо производить поиск в подпапках
`Execute`	Выполняет поиск
`NewSearch`	Сбрасывает состояние объекта `FileSearch`

Далее вы ознакомитесь с примерами использования указанных методов и свойств объекта `FileSearch`.

ОТОБРАЖЕНИЕ СПИСКА ФАЙЛОВ В ПАПКЕ

Следующая процедура отображает (на активном рабочем листе) список файлов, которые находятся в указанной папке. Кроме того, указывается размер файлов и время их последней модификации.

```
Sub ListFiles2()
    Directory = "c:\windows\"

'   Вставка заголовков
    r = 1
    Cells.ClearContents
    Cells(r, 1) = "Имя файла"
    Cells(r, 2) = "Размер"
```

```
    Cells(r, 3) = "Дата/время"
    Range("A1:C1").Font.Bold = True
    r = r + 1

    On Error Resume Next
    With Application.FileSearch
        .NewSearch
        .LookIn = Directory
        .Filename = "*.*"
        .SearchSubFolders = False
        .Execute
        For i = 1 To .FoundFiles.Count
            Cells(r, 1) = .FoundFiles(i)
            Cells(r, 2) = FileLen(.FoundFiles(i))
            Cells(r, 3) = FileDateTime(.FoundFiles(i))
            r = r + 1
        Next i
    End With
End Sub
```

Объект `FileSearch` игнорирует все ярлыки Windows (файлы с расширением `*.LNK`).

ОПРЕДЕЛЕНИЕ ФАКТА СУЩЕСТВОВАНИЯ ФАЙЛА

Приведенная ниже функция принимает два аргумента (путь и имя файла) и возвращает значение `True`, если файл существует в указанной папке. После запуска метода `Execute` свойство `Count` объекта `FoundFiles` будет равно значению 1, если файл найден.

```
Function FileExists2(path, fname) As Boolean
    With Application.FileSearch
        .NewSearch
        .Filename = fname
        .LookIn = path
        .Execute
        If .FoundFiles.Count = 1 Then
            FileExists2 = True
        Else
            FileExists2 = False
        End If
    End With
End Function
```

Объект `FileSearch` невозможно использовать для определения факта существования папки.

Использование объекта FileSystemObject

Объект `FileSystemObject` является членом библиотеки Windows Scripting Host и предоставляет доступ к файловой системе компьютера. Этот объект часто применяется в ориентированных на использование сценариев Web-приложениях (например, использующих код VBScript или JavaScript) и поддерживается в Excel 2000 и в более поздних версиях программы.

Библиотека Windows Scripting Host часто применяется для распространения компьютерных вирусов. Следовательно, Windows Scripting Host отключена в большинстве систем. Таким образом, очень внимательно относитесь к разработке приложения, которое будет использоваться в различных системах.

Документацию к объекту `FileSystemObject` вы найдете по адресу `http://msdn.microsoft.com/scripting`.

ОПРЕДЕЛЕНИЕ ФАКТА СУЩЕСТВОВАНИЯ ФАЙЛА

Функция, которая приводится ниже, принимает один аргумент (путь и имя файла), после чего возвращает значение `True`, если указанный файл существует.

```
Function FileExists3(fname) As Boolean
    Set FileSys = CreateObject("Scripting.FileSystemObject")
    FileExists3 = FileSys.FileExists(fname)
End Function
```

Функция создает новый объект `FileSystemObject`, который называется `FileSys`, и получает доступ к свойству `FileExists` этого объекта.

На прилагаемом к книге компакт-диске содержится пример, который демонстрирует все три метода (команды VBA, объект `FileSearch` и объект `FileSystemObject`) определения факта существования файла.

ОПРЕДЕЛЕНИЕ ФАКТА СУЩЕСТВОВАНИЯ ПАПКИ

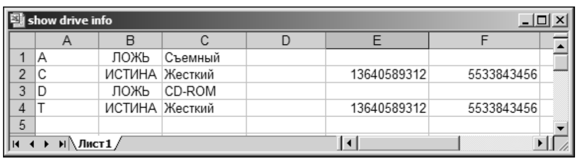
Функция, которая приводится ниже, принимает один аргумент (путь) и возвращает значение `True`, если указанная папка существует.

```
Function PathExists2(path) As Boolean
    Set FileSys = CreateObject("Scripting.FileSystemObject")
    On Error Resume Next
    Set FolderObj = FileSys.getfolder(path)
    If Err = 0 Then
        PathExists2 = True
    Else
        PathExists2 = False
    End If
End Function
```

С помощью данной функции создается новый объект `Folder`, который называется `FolderObj`. Если операция завершается успешно, то папка существует. Если возникает сообщение об ошибке, то папка на диске отсутствует.

ПОЛУЧЕНИЕ ИНФОРМАЦИИ О ДИСКАХ

Следующая процедура использует объект `FileSystemObject` для отображения различной информации о дисках. Процедура циклически просматривает коллекцию `Drives` и записывает значения свойств на рабочий лист. На рис. 27.2 показаны результаты выполнения процедуры в системе, в которой установлен дисковод чтения гибких дисков, несколько жестких дисков и два устройства чтения компакт-дисков. Отображается информация о названии диска (буква), состоянии "готовности" диска, типе диска, имени тома диска, общем объеме, а также объеме свободного пространства.



	A	B	C	D	E	F
1	A	ЛОЖЬ	Съемный			
2	C	ИСТИНА	Жесткий		13640589312	5533843456
3	D	ЛОЖЬ	CD-ROM			
4	T	ИСТИНА	Жесткий		13640589312	5533843456
5						

Рис. 27.2. Результат выполнения процедуры `ShowDriveInfo`

В некоторых версиях Windows свойства `TotalSize` и `AvailableSpace` могут возвращать некорректные значения для дисков, емкость которых больше 2 Гбайт. Данная проблема обойдена в Windows NT, а также Windows 2000 и более поздних версиях ОС.

Эта рабочая книга доступна на прилагаемом к книге компакт-диске.

```
Sub ShowDriveInfo()
    Dim FileSys, Drv
    Dim Row As Integer

    Set FileSys = CreateObject("Scripting.FileSystemObject")
    Cells.Clear
    Row = 0
    On Error Resume Next
    For Each Drv In FileSys.Drives
        Row = Row + 1
        Cells(Row, 1) = Drv.DriveLetter
        Cells(Row, 2) = Drv.IsReady
        Select Case Drv.DriveType
            Case 0: Cells(Row, 3) = "Неизвестно"
            Case 1: Cells(Row, 3) = "Съемный"
            Case 2: Cells(Row, 3) = "Жесткий"
            Case 3: Cells(Row, 3) = "Сетевой"
            Case 4: Cells(Row, 3) = "CD-ROM"
            Case 5: Cells(Row, 3) = "RAM"
        End Select
        Cells(Row, 4) = Drv.VolumeName
        Cells(Row, 5) = Drv.TotalSize
        Cells(Row, 6) = Drv.AvailableSpace
    Next Drv
End Sub
```

Поиск файлов, которые содержат определенный текст

Следующая процедура производит поиск файлов `*.xls` в папке `Мои документы` и ее подпапках, которые содержат текст `бюджет`. Все найденные имена файлов добавляются в список элемента управления `ListBox` диалогового окна `UserForm`.

```
Sub FindFiles()
    With Application.FileSearch
        .NewSearch
        .LookIn = "C:\Мои документы"
        .SearchSubFolders = True
        .TextOrProperty = "бюджет"
        .MatchTextExactly = False
        .Filename = "*.xls"
        .Execute
        For i = 1 To .FoundFiles.Count
            UserForm1.ListBox1.AddItem .FoundFiles(i)
        Next i
    End With
    UserForm1.Show
End Sub
```

Работа с текстовыми файлами

VBA содержит ряд операторов, которые позволяют управлять файлами на низком уровне. Эти операторы ввода/вывода предоставляют разработчику более широкие возможности, чем стандартные средства экспорта и импорта файлов Excel.

Доступ к файлу можно осуществлять в одном из трех режимов.

- *Последовательный доступ.* Этот режим является самым распространенным. Он позволяет считывать и записывать отдельные символы или целые строки данных.
- *Произвольный доступ.* Режим, используемый только при программировании приложений баз данных (не рекомендуется использовать в VBA, поскольку для этого существуют более подходящие средства).
- *Двоичный доступ.* Используется для чтения и записи с любой позиции в файле с точностью до байта. Данный режим может применяться для сохранения или отображения растрового изображения. В VBA он практически не используется.

По причине того, что произвольный и двоичный режимы доступа в VBA используются редко, в данной главе основное внимание будет уделено рассмотрению последовательного доступа. При последовательном доступе файл считывается последовательно. Другими словами, приложение начинает чтение с начала файла и последовательно получает строку за строкой. При записи файла приложение добавляет строки в конец существующего файла.

Для чтения и записи файлов в этой главе используется традиционный метод "канала данных". Еще одной возможностью является применение "объектного" подхода. Объект `FileSystemObject` содержит объект `TextStream`, который может использоваться для чтения и записи текстовых файлов. Объект `FileSystemObject` является частью библиотеки Windows Scripting Host. Как отмечалось ранее, такая служба поддержки сценариев отключена в большинстве систем из-за ее предрасположенности к распространению вирусов.

Открытие текстового файла

Оператор VBA `Open` (не путайте с методом `Open` объекта `Application`) используется при открытии файла для чтения или записи. Перед тем, как начать чтение или запись файла, его необходимо открыть.

Оператор `Open` является очень гибким, поэтому он имеет относительно сложный синтаксис.

```
Open путь For режим [Access доступ] [блокировка] _
  As [#]номер_файла [Len=дл_записи]
```

- *`путь`* — аргумент *`путь`* оператора `Open` очень простой. Он указывает имя и путь файла, который необходимо открыть (обязательный параметр).
- *`режим`* — это режим, в котором должен открываться файл (обязательный параметр). Данный параметр может иметь следующие значения:
  - `Append` — режим последовательного доступа, который позволяет осуществлять чтение файла, а также добавлять данные в конец файла;
  - `Input` — режим последовательного доступа, который позволяет читать файл, но не дает возможности записывать в него данные;

- `Output` — режим последовательного доступа, который позволяет выполнять чтение и запись файла. В этом режиме всегда создается новый файл (существующий файл с текущим именем удаляется);
- `Binary` — режим произвольного доступа, в котором производится побайтовое чтение и запись данных;
- `Random` — режим произвольного доступа, который позволяет выполнять чтение и запись данных блоками, установленными значением аргумента *дл_записи* оператора `Open`.

♦ *доступ* — этот аргумент определяет допустимые операции над файлом (необязательный параметр). Этот параметр может иметь значения `Read`, `Write` и `Read Write`.

♦ *блокировка* — аргумент *блокировка* используется для разрешения проблем многопользовательского доступа (необязательный параметр). Этот параметр может иметь следующие значения: `Shared`, `Lock Read`, `Lock Write`, а также `Lock Read Write`.

♦ *номер_файла* — номер файла, который находится в пределах от 1 до 511 (обязательный параметр). Для получения следующего свободного номера файла можно использовать функцию `FreeFile`.

♦ *дл_записи* — длина записи (для файлов с произвольным доступом) или размер буфера (для файлов с последовательным доступом) — необязательный параметр.

Чтение текстового файла

Базовой процедурой чтения текстового файла с использованием VBA является следующая последовательность действий.

1. Открытие файла с помощью оператора `Open`.
2. Указание позиции в файле с помощью функции `Seek` (не обязательно).
3. Чтение данных из файла с помощью функций `Input`, `Input #` или `Line Input #`.
4. Закрытие файла с помощью оператора `Close`.

Запись в текстовый файл

Базовая процедура записи текстового файла состоит из следующей последовательности действий.

1. Открытие или создание файла с помощью оператора `Open`.
2. Указание позиции в файле с помощью функции `Seek` (не обязательно).
3. Запись данных в файл с помощью операторов `Write #` или `Print #`.
4. Закрытие файла с помощью оператора `Close`.

Получение номера файла

Большинство программистов, использующих VBA, просто назначают номер файла в вызове оператора `Open`.

```
Open "myfile.txt" For Input As #1
```

После этого в следующих операторах на файл можно ссылаться как на #1.

Если открыт первый файл и вы открываете второй, то последний получит номер #2.

```
Open "another.txt" For Input As #2
```

Средства Excel импорта и экспорта текстовых файлов

Excel поддерживает три типа текстовых файлов.

- *Файлы CSV (Comma Separated Values) (Разделенные запятыми значения)*. Столбцы данных разделяются запятыми, а каждая строка данных завершается символом возврата каретки. В некоторых неанглийских версиях Excel вместо запятой используется точка с запятой.
- *PRN*. Столбцы данных выравниваются по определенному символу, а каждая строка завершается символом возврата каретки.
- *Файлы TXT (Tab-delimited) (Разделенные символами табуляции)*. Столбцы данных разделяются с помощью символов табуляции, а каждая строка данных завершается символом возврата каретки.

При попытке открыть текстовый файл с помощью команды Файл⇨Открыть будет отображен мастер импорта текстовых файлов, который поможет указать пользователю способ разделения столбцов. Если в тексте в качестве разделителя применялся символ табуляции или запятая, то Excel откроет файл без помощи мастера импорта текстовых файлов. Мастер преобразования текста в столбцы, доступ к которому осуществляется с помощью команды Данные⇨Текст по столбцам, является аналогом мастера импорта текстовых файлов, но работает с данными, которые представлены единственным столбцом.

Еще одним методом является использование функции VBA `FreeFile`, которая позволяет получить свободный дескриптор файла. После этого на файл можно ссылаться с помощью переменной. Приведем пример использования этой функции.

```
FileHadle = FreeFile
Open "myfile.txt" For Input As FileHandle
```

Определение или установка позиции в файле

Для последовательного доступа к файлу редко когда возникает необходимость в получении текущей позиции. Но если по определенной причине такая информация нужна, то можно воспользоваться функцией `Seek`.

Операторы чтения и записи

VBA предоставляет несколько операторов чтения и записи данных в файл.

Представленные далее операторы используются для чтения данных из файла с последовательным доступом.

- `Input` — читает из файла указанное количество символов.
- `Input #` — читает файл в виде последовательности переменных; переменные разделяются запятой.
- `Line Input #` — читает файл построчно (строки разделяются символами возврата каретки и/или перевода строки).

Для записи данных в файл с последовательным доступом используются два оператора.

- Write # — записывает последовательность значений, в которой каждое значение отделено запятой и находится в кавычках. Если оператор завершается точкой с запятой, после каждого значения не будет вставляться последовательность возврат каретки/перевод строки. Данные, записанные с помощью оператора Write #, обычно читаются из файла с помощью оператора Input #.
- Print # — записывает последовательность значений, в которой каждое значение отделено символом пробела. Если завершить оператор точкой с запятой, то после каждого значения не будет вставляться последовательность возврат каретки/перевод строки. Данные, записанные с помощью оператора Print #, обычно читаются с помощью оператора Line Input # или оператора Input.

Примеры управления текстовыми файлами

В данном разделе содержится несколько примеров, демонстрирующих различные способы управления текстовыми файлами.

Импортирование данных из текстового файла

Следующий пример читает текстовый файл и размещает каждую строку данных в отдельную ячейку (начиная с активной ячейки).

```
Sub ImportData()
    Set ImpRng = ActiveCell
    Open "c:\windows\desktop\textfile.txt" For Input As #1
    r = 0
    Do Until EOF(1)
        Line Input #1, data
        ActiveCell.Offset(r, 0) = data
        r = r + 1
    Loop
    Close #1
End Sub
```

В большинстве случаев данная процедура не принесет пользы, так как каждая строка данных просто помещается в отдельную ячейку. Но можно воспользоваться командой Данные⇨Текст по столбцам для разбора данных по столбцам.

Экспортирование диапазона в текстовый файл

В приведенном далее примере (листинг 27.3) демонстрируется простая процедура, которая записывает данные из выделенного диапазона на рабочем листе в текстовый файл в формате CSV.

Обратите внимание на то, что процедура использует два оператора Write #. Первый завершается точкой с запятой, поэтому последовательность возврат каретки/перевод строки в файл не добавляется. Но для последней ячейки в строке второй оператор Write # не имеет точки с запятой, что приводит к завершению строки и добавлению следующей ячейки уже в новой строке.

Переменная Data используется для хранения содержимого каждой ячейки. Если ячейка имеет числовой формат, то переменная принимает значение, что обеспечивает отсутствие кавычек при сохранении данных. Если ячейка пуста, то ее свойство Value возвращает значение 0. Таким образом, проверяются пустые ячейки (для этого используется функция IsEmpty) и вместо 0 подставляется пустая строка.

Листинг 27.3. Сохранение данных выделенного диапазона в текстовый CSV-файл

```
Sub ExportRange()
    Dim Filename As String
    Dim NumRows As Long, NumCols As Integer
    Dim r As Long, c As Integer
    Dim Data
    Dim ExpRng As Range
    Set ExpRng = Selection
    NumCols = ExpRng.Columns.Count
    NumRows = ExpRng.Rows.Count
    Filename = "c:\windows\textfile.txt"
    Open Filename For Output As #1
        For r = 1 To NumRows
            For c = 1 To NumCols
                Data = ExpRng.Cells(r, c).Value
                If IsNumeric(Data) Then Data = Val(Data)
                If IsEmpty(ExpRng.Cells(r, c)) Then Data = ""
                If c <> NumCols Then
                    Write #1, Data;
                Else
                    Write #1, Data
                End If
            Next c
        Next r
    Close #1
End Sub
```

Этот пример доступен на прилагаемом к книге компакт-диске.

На рис. 27.3 показано содержимое результирующего файла.

Импортирование текстового файла в диапазон

Представленная далее подпрограмма (рис. 27.4) читает содержимое текстового файла, который был создан в предыдущем примере. После этого прочитанные значения сохраняются, начиная с активной ячейки. Приложение читает каждый символ и разбирает полученные строки данных. При этом символы кавычек игнорируются, а запятые используются для разделения столбцов.

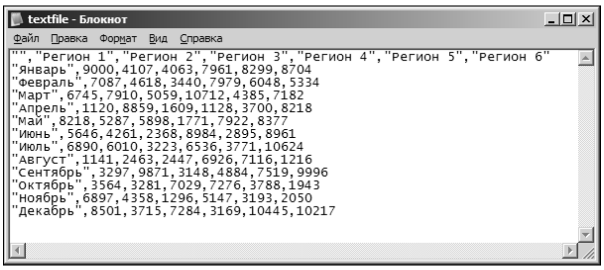


Рис. 27.3. Текстовый файл, который создан с помощью кода VBA

Листинг 27.4. Чтение файла предыдущего листинга и занесение данных, начиная с активной ячейки

```
Sub ImportRange()
    Dim ImpRng As Range
    Dim Filename As String
    Dim r As Long, c As Integer
    Dim txt As String, Char As String * 1
    Dim Data
    Dim i As Integer

    Set ImpRng = ActiveCell
    On Error Resume Next
    Filename = "c:\windows\textfile.txt"
    Open Filename For Input As #1
    If Err <> 0 Then
        MsgBox "Невозможно найти: " & Filename, vbCritical, "ОШИБКА"
        Exit Sub
    End If
    r = 0
    c = 0
    txt = ""
    Application.ScreenUpdating = False
    Do Until EOF(1)
        Line Input #1, Data
        For i = 1 To Len(Data)
            Char = Mid(Data, i, 1)
            If Char = "," Then 'запятая
                ActiveCell.Offset(r, c) = txt
                c = c + 1
                txt = ""
            ElseIf i = Len(Data) Then 'конец строки
                If Char <> Chr(34) Then txt = txt & Char
                ActiveCell.Offset(r, c) = txt
                txt = ""
            ElseIf Char <> Chr(34) Then
                txt = txt & Char
            End If
        Next i
        c = 0
        r = r + 1
    Loop
    Close #1
    Application.ScreenUpdating = True
End Sub
```

Процедура, показанная выше, имеет недостаток: она не поддерживает значения, которые содержат символы кавычек или запятой. Кроме того, импортированные даты будут окружены символами номера, например: #2001-05-12#.

Данный пример доступен на прилагаемом к книге компакт-диске.

Протоколирование операций в Excel

Пример, приведенный в этом разделе, предназначен для записи данных в текстовый файл при каждом открытии и закрытии Excel. Для того чтобы эта процедура работала надежно, она должна находиться в рабочей книге, которая всегда открывается и закрывается вместе с Excel. Рекомендуется сохранять эту процедуру в персональной книге макросов.

Следующая процедура, которая находится в модуле кода объекта `ЭтаКнига`, выполняется каждый раз при открытии файла.

```
Private Sub Workbook_Open()
    Open Application.Path & "\excelusage.txt" _
      For Append As #1
    Print #1, "Started " & Now
    Close #1
End Sub
```

Процедура добавляет новую строку в файл, который называется `excelusage.txt`. Новая строка содержит текущую дату и время и может выглядеть следующим образом:

```
Started 03/09/00 9:27:43 PM
```

Представленная ниже подпрограмма выполняется каждый раз при закрытии рабочей книги. Эта процедура добавляет к текстовому файлу новую строку, которая содержит слово "Stopped", а также текущее время и дату.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Open Application.Path & "\excelusage.txt" _
      For Append As #1
    Print #1, "Stopped " & Now
    Close #1
End Sub
```

Фильтрация текстового файла

Пример данного раздела демонстрирует одновременную работу с двумя текстовыми файлами. Процедура `FilterFile`, которая приводится ниже, читает текстовый файл (`infile.txt`) и копирует строки, содержащие определенный текст, во второй текстовый файл (`output.txt`).

```
Sub FilterFile()
    Open "infile.txt" For Input As #1
    Open "output.txt" For Output As #2
    TextToFind = "January"
    Do Until EOF(1)
        Line Input #1, data
        If InStr(1, data, TextToFind) Then
            Print #2, data
        End If
    Loop
    Close
End Sub
```

Этот пример доступен на прилагаемом к книге компакт-диске.

Импортирование более 256 столбцов данных

Часто возникает необходимость в импортировании данных, которые превышают ограничение Excel в 256 столбцов. Если попытаться открыть такой файл с помощью команды Файл⇨Открыть, то Excel просто проигнорирует данные, которые находятся за пределами 256-го столбца (при этом предупреждение на экране не отображается)!

Следующая процедура (листинг 27.5) является вариантом процедуры `ImportRange`, которая приводилась ранее в этой главе. Такая процедура выполняет чтение текстового файла и импортирует данные в новую рабочую книгу. Если данные содержат более 256-ти столбцов, то в рабочую книгу добавляются дополнительные листы.

Листинг 27.5. Чтение текстового файла и импортирование его в новую рабочую книгу

```
Sub ImportLongLines()
'   Импорт текстового файла объемом более 256 столбцов

    Dim ImpRange As Range
    Dim r As Long, c As Integer
    Dim CurrLine As Long
    Dim Data As String, Char As String, Txt As String
    Dim i As Integer
    Dim CurrSheet As Worksheet

'   Создание рабочей книги с одним листом
    Workbooks.Add xlWorksheet

    Open ThisWorkbook.Path & "\longfile.txt" For Input As #1
    r = 0
    c = 0
    Set ImpRange = ActiveWorkbook.Sheets(1).Range("A1")
    Application.ScreenUpdating = False

'   Чтение первой строки, вставка листа (если требуется)
    CurrLine = CurrLine + 1
    Line Input #1, Data
    For i = 1 To Len(Data)
        Char = Mid(Data, i, 1)
'       Столбцы закончились?
        If c <> 0 And c Mod 256 = 0 Then
            Set CurrSheet = ActiveWorbook.Sheets.Add(after:=ActiveWorkbook. _
Sheets(ActiveWorkbook.Sheets.Count))
            Set ImpRange = CurrSheet.Range("A1")
            c = 0
        End If
'       Конец поля?
        If Char = "," Then
            ImpRange.Offset(r, c) = Txt
            c = c + 1
            Txt = ""
        Else
'           Пропуск символов кавычек
            If Char <> Chr(34) Then _
                Txt = Txt & Mid(Data, i, 1)

'           Конец строки?
            If i = Len(Data) Then
                ImpRange.Offset(r, c) = Txt
                c = c + 1
                Txt = ""
            End If
        End If
    Next i

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'   Чтение остальных данных
    c = 0
    CurrLine = 1
    Set ImpRange = ActiveWorkbook.Sheets(1).Range("A1")
    r = r + 1

    Do Until EOF(1)
        Set ImpRange = ActiveWorkbook.Sheets(1).Range("A1")
        CurrLine = CurrLine + 1
        Line Input #1, Data
        Application.StatusBar = "Обработка строки " & CurrLine
```

```
        For i = 1 To Len(Data)
            Char = Mid(Data, i, 1)
'           Столбцы закончились?
            If c <> 0 And c Mod 256 = 0 Then
                c = 0
                Set ImpRange = ImpRange.Parent.Next.Range("A1")
            End If

'           Конец поля
            If Char = "," Then
                ImpRange.Offset(r, c) = Txt
                c = c + 1
                Txt = ""
            Else
'               Пропуск символов кавычек
                If Char <> Chr(34) Then _
                    Txt = Txt & Mid(Data, i, 1)

'               Конец строки?
                If i = Len(Data) Then
                    ImpRange.Offset(r, c) = Txt
                    c = c + 1
                    Txt = ""
                End If
            End If
        Next i
        c = 0
        Set ImpRange = ActiveWorkbook.Sheets(1).Range("A1")
        r = r + 1
    Loop

'   Очистка
    Close #1
    Application.ScreenUpdating = True
    Application.StatusBar = False
End Sub
```

Эта процедура состоит из двух частей. Первая часть читает первую строку данных и добавляет новые рабочие листы, если это необходимо. Вторая часть читает оставшиеся строки текстового файла. Код предполагает, что первая строка имеет тот же формат, что и оставшаяся часть данных, и что первая строка характеризуется максимальным числом столбцов.

НА КОМПАКТ-ДИСКЕ

Этот пример доступен на прилагаемом к книге компакт-диске. Более того, вместе с примером предоставляется текстовый файл, содержащий 100 строк и 600 столбцов данных.

Экспортирование диапазона в формат HTML

Предпоследний пример данной главы демонстрирует операцию экспорта диапазона ячеек в файл формата HTML. Файл HTML, как легко догадаться, является текстовым файлом, содержащим специальные дескрипторы форматирования, которые определяют внешний вид данных при отображении в браузере.

Что мешает использовать команду Excel Файл⇨Сохранить как Web страницу? Процедура, приведенная в данном примере, имеет одно заметное преимущество: она не создает “раздутый” код HTML. Например, процедура `ExportToHTML` использовалась для экспорта диапазона из 70 ячеек. Результирующий файл имеет размер 2,7 Кбайт. Если выполнить команду Excel Файл⇨Сохранить как Web страницу, то результирующий файл будет иметь размер 15,8 Кбайт (в шесть раз больше). С другой стороны, процедура `ExportToHTML` не сохраняет все параметры форматирования ячеек. Она

поддерживает только форматирование с помощью полужирного и курсивного начертания. Кроме того, процедура ExportToHTML имеет еще один серьезный недостаток: она не поддерживает объединенные ячейки.

Ниже приведен исходный код процедуры ExportToHTML.

Листинг 27.6. Экспортирование диапазона в HTML-файл

```
Sub ExportToHTML()
'    Dim ws As Worksheet
    Dim Filename As Variant
    Dim TDOpenTag As String, TDCloseTag As String
    Dim CellContents As String
    Dim Rng As Range
    Dim r As Long, c As Integer

'   Использование выделенного диапазона ячеек
    Set Rng = Application.Intersect(ActiveSheet.UsedRange, Selection)

'   Получение имени файла
    Filename = Application.GetSaveAsFilename( _
        InitialFileName:="myrange.htm", _
        fileFilter:="HTML Files(*.htm), *.htm")
    If Filename = False Then Exit Sub

'   Открытие текстового файла
    Open Filename For Output As #1

'   Запись дескриптора <TABLE>
    Print #1, "<TABLE BORDER=1 CELLPADDING=3>"

'   Циклический просмотр ячеек
    For r = 1 To Rng.Rows.Count
        Print #1, "<TR>"
        For c = 1 To Rng.Columns.Count
            TDOpenTag = "<TD ALIGN=RIGHT>"
            TDCloseTag = "</TD>"
            If Rng.Cells(r, c).Font.Bold Then
                TDOpenTag = TDOpenTag & "<B>"
                TDCloseTag = "</B>" & TDCloseTag
            End If
            If Rng.Cells(r, c).Font.Italic Then
                TDOpenTag = TDOpenTag & "<I>"
                TDCloseTag = "</I>" & TDCloseTag
            End If
            CellContents = Rng.Cells(r, c).Text
            Print #1, TDOpenTag & CellContents & TDCloseTag
        Next c
        Print #1, "</TR>"
    Next r
'   Закрытие таблицы
    Print #1, "</TABLE>"

'   Закрытие файла
    Close #1

'   Создание сообщения пользователю
    MsgBox Rng.Count & " ячеек экспортировано в файл " & Filename
End Sub
```

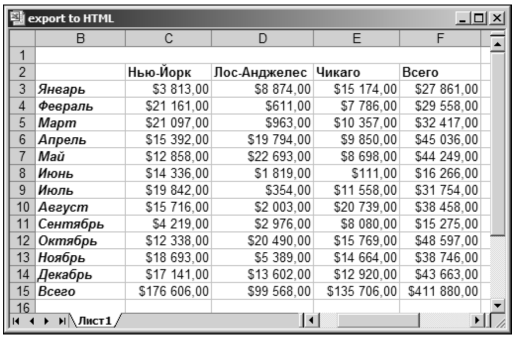
НА КОМПАКТ-ДИСКЕ

Этот пример доступен на прилагаемом к книге компакт-диске.

Процедура начинает свою работу с определения экспортируемого диапазона. Данный диапазон определяется как пересечение выделенного диапазона и используемой области рабочего листа. Это позволяет удостовериться, что не будет обрабатываться весь столбец или строка. Пользователь видит на экране диалоговое окно с просьбой указать имя файла. Далее открывается указанный текстовый файл. Основная работа выполняется в двух циклах `For-Next`. Код генерирует соответствующий код HTML (записывает необходимые дескрипторы) и заносит в файл экспортируемые данные. Наконец, файл закрывается, и пользователь может увидеть окно сообщения с итоговой информацией.

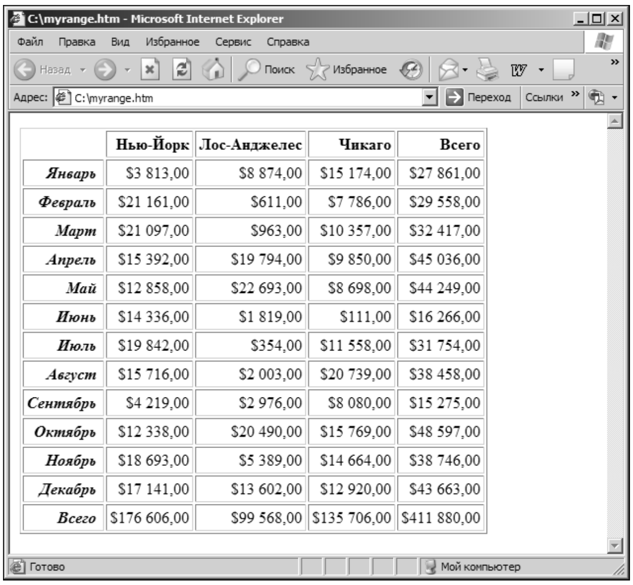
На рис. 27.4 показан диапазон рабочего листа, а на рис. 27.5 представлено, как этот диапазон выглядит в браузере после экспорта данных в формат HTML.

Процедуру `ExportToHTML` можно использовать в качестве основы для последующего изменения текстовых данных.



	B	C	D	E	F
1					
2		Нью-Йорк	Лос-Анджелес	Чикаго	Всего
3	Январь	$3 813,00	$8 874,00	$15 174,00	$27 861,00
4	Февраль	$21 161,00	$611,00	$7 786,00	$29 558,00
5	Март	$21 097,00	$963,00	$10 357,00	$32 417,00
6	Апрель	$15 392,00	$19 794,00	$9 850,00	$45 036,00
7	Май	$12 858,00	$22 693,00	$8 698,00	$44 249,00
8	Июнь	$14 336,00	$1 819,00	$111,00	$16 266,00
9	Июль	$19 842,00	$354,00	$11 558,00	$31 754,00
10	Август	$15 716,00	$2 003,00	$20 739,00	$38 458,00
11	Сентябрь	$4 219,00	$2 976,00	$8 080,00	$15 275,00
12	Октябрь	$12 338,00	$20 490,00	$15 769,00	$48 597,00
13	Ноябрь	$18 693,00	$5 389,00	$14 664,00	$38 746,00
14	Декабрь	$17 141,00	$13 602,00	$12 920,00	$43 663,00
15	Всего	$176 606,00	$99 568,00	$135 706,00	$411 880,00
16					

Рис. 27.4. Диапазон рабочего листа, который будет экспортирован в формат HTML



	Нью-Йорк	Лос-Анджелес	Чикаго	Всего
Январь	$3 813,00	$8 874,00	$15 174,00	$27 861,00
Февраль	$21 161,00	$611,00	$7 786,00	$29 558,00
Март	$21 097,00	$963,00	$10 357,00	$32 417,00
Апрель	$15 392,00	$19 794,00	$9 850,00	$45 036,00
Май	$12 858,00	$22 693,00	$8 698,00	$44 249,00
Июнь	$14 336,00	$1 819,00	$111,00	$16 266,00
Июль	$19 842,00	$354,00	$11 558,00	$31 754,00
Август	$15 716,00	$2 003,00	$20 739,00	$38 458,00
Сентябрь	$4 219,00	$2 976,00	$8 080,00	$15 275,00
Октябрь	$12 338,00	$20 490,00	$15 769,00	$48 597,00
Ноябрь	$18 693,00	$5 389,00	$14 664,00	$38 746,00
Декабрь	$17 141,00	$13 602,00	$12 920,00	$43 663,00
Всего	$176 606,00	$99 568,00	$135 706,00	$411 880,00

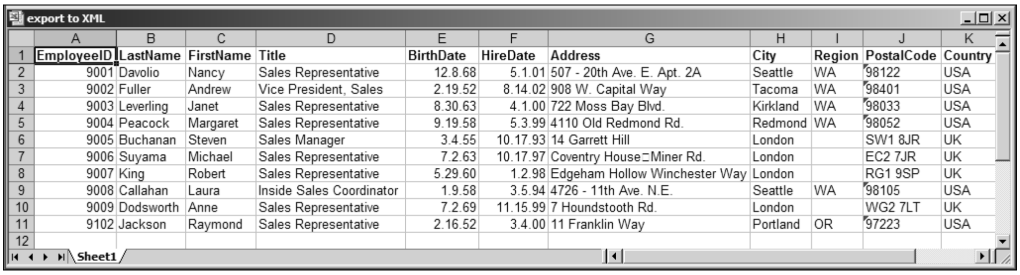
Рис. 27.5. Данные рабочего листа, экспортированные в формат HTML

Экспортирование диапазона в XML-файл

В последнем примере этой главы диапазон данных Excel экспортируется в XML-файл. Как вы знаете, в XML каждый элемент данных помечается собственным дескриптором. Процедура этого примера использует в качестве дескрипторов подписи первой строки. На рис. 27.6 показан исходный диапазон, а на рис. 27.7 — готовый XML-файл, отображенный в Internet Explorer.



Excel 2003 поддерживает технологию XML, однако с ее помощью вы вряд ли создадите XML-файлы на основе диапазона данных без использования файла карты (схемы) XML.



EmployeeID	LastName	FirstName	Title	BirthDate	HireDate	Address	City	Region	PostalCode	Country
9001	Davolio	Nancy	Sales Representative	12.8.68	5.1.01	507 - 20th Ave. E. Apt. 2A	Seattle	WA	98122	USA
9002	Fuller	Andrew	Vice President, Sales	2.19.52	8.14.02	908 W. Capital Way	Tacoma	WA	98401	USA
9003	Leverling	Janet	Sales Representative	8.30.63	4.1.00	722 Moss Bay Blvd.	Kirkland	WA	98033	USA
9004	Peacock	Margaret	Sales Representative	9.19.58	5.3.99	4110 Old Redmond Rd.	Redmond	WA	98052	USA
9005	Buchanan	Steven	Sales Manager	3.4.55	10.17.93	14 Garrett Hill	London		SW1 8JR	UK
9006	Suyama	Michael	Sales Representative	7.2.63	10.17.97	Coventry House Miner Rd.	London		EC2 7JR	UK
9007	King	Robert	Sales Representative	5.29.60	1.2.98	Edgeham Hollow Winchester Way	London		RG1 9SP	UK
9008	Callahan	Laura	Inside Sales Coordinator	1.9.58	3.5.94	4726 - 11th Ave. N.E.	Seattle	WA	98105	USA
9009	Dodsworth	Anne	Sales Representative	7.2.69	11.15.99	7 Houndstooth Rd.	London		WG2 7LT	UK
9102	Jackson	Raymond	Sales Representative	2.16.52	3.4.00	11 Franklin Way	Portland	OR	97223	USA

Рис. 27.6. Исходные данные, которые преобразуются в формат XML

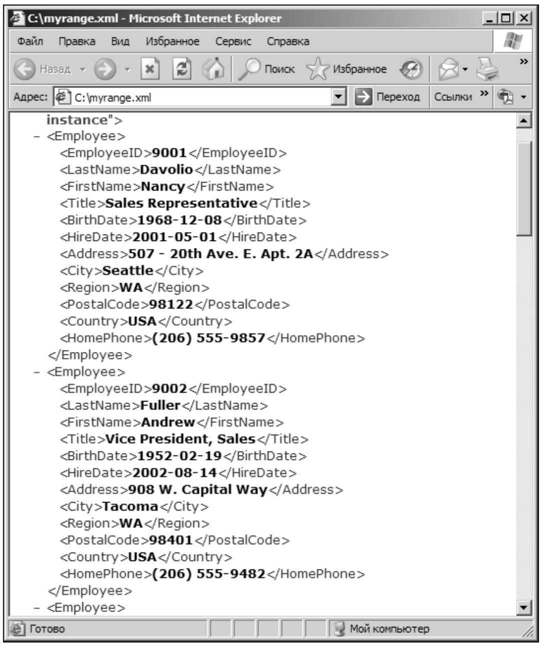


Рис. 26.7. Конечный файл в браузере

Приведенная ниже процедура имеет много общего с процедурой ExportToHTML предыдущего примера.

Листинг 27.7. Экспортирование диапазона данных в формат XML

```
Sub ExportToXML()
'   Dim ws As Worksheet
    Dim Filename As Variant
    Dim TDOpenTag As String, TDCloseTag As String
    Dim CellContents As String
    Dim Rng As Range
    Dim r As Long, c As Integer

'   Назначение диапазона
    Set Rng = Range("A1:L11")

'   Получение имени файла
    Filename = Application.GetSaveAsFilename( _
        InitialFileName:="myrange.xml", _
        fileFilter:="XML Files(*.xml), *.xml")
    If Filename = False Then Exit Sub

'   Открытие текстового файла
    Open Filename For Output As #1

'   Создание дескриптора <xml>
    Print #1, "<?xml version=""1.0"" encoding=""UTF-8"" standalone=""yes""?>"
    Print #1, "<EmployeeList xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance"">"

'   Просмотр всех ячеек
    For r = 2 To Rng.Rows.Count
        Print #1, "<Employee>"
        For c = 1 To Rng.Columns.Count
            Print #1, "<" & Rng.Cells(1, c) & ">";
            If IsDate(Rng.Cells(r, c)) Then
                Print #1, Format(Rng.Cells(r, c), "yyyy-mm-dd");
            Else
                Print #1, Rng.Cells(r, c).Text;
            End If
            Print #1, "</" & Rng.Cells(1, c) & ">"
        Next c
        Print #1, "</Employee>"
    Next r
'   Закрытие таблицы
    Print #1, "</EmployeeList>"

'   Закрытие файла
    Close #1

'   Сообщение пользователю
    MsgBox Rng.Rows.Count - 1 & " ячеек экспортировано в файл " & Filename
End Sub
```

Глава 28

Управление компонентами Visual Basic

В ЭТОЙ ГЛАВЕ...

В данной главе рассматривается тема, которая может оказаться довольно важной для многих из вас, — создание кода VBA для управления компонентами проектов.

- Обзор интегрированной среды разработки.
- Важная информация для пользователей Excel 2002–2003.
- Использование VBA для добавления и удаления модулей проекта.
- Создание кода VBA для создания другого кода VBA.
- Использование кода VBA для упрощенного создания диалоговых окон `UserForm`.
- Полезная функция создания диалоговых окон `UserForm`.

Интегрированная среда разработки VBA содержит объектную модель, которая содержит ключевые элементы проекта VBA, включая сам редактор. В результате можно создать код VBA, который добавляет или удаляет модули, создает дополнительный код VBA или даже диалоговые окна `UserForm`.

Введение в IDE

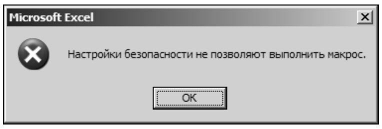
IDE — это интегрированная среда разработки OLE для редактора Visual Basic Editor. Сразу после создания ссылки на библиотеку Visual Basic Extensibility Library (с помощью команды Tools⇨References (Сервис⇨Ссылки)) разработчику предоставляется доступ ко всем объектам, свойствам и методам VBE, а также позволено объявлять объекты из классов-членов IDE.

В диалоговом окне References выберите опцию Microsoft Visual Basic for Application Extensibility. Это предоставит доступ к объекту `VBIDE`. Создание ссылки на объект `VBIDE` позволит объявлять объекты, которые входят в состав `VBIDE`, а также открывает доступ к константам, определенным внутри IDE. Отметим, что доступ к объектам IDE можно получать и *не создавая ссылку*, но при этом у вас не будет возможности использовать константы IDE, а также объявлять объекты, которые ссылаются на компоненты IDE.

Важные замечания для пользователей Excel 2002–2003

Если для разработки приложений используется Excel 2002–2003, то необходимо помнить, что в Excel 2002 многое изменилось. Для того чтобы уменьшить вероятность заражения макровирусами, Microsoft усложнила процедуры использования VBA для модификации компонентов проекта VBA. Если попытаться запустить одну из процедур, приводимых в этой главе, то на экране может появиться сообщение об ошибке, показанное ниже.

Отображение этого сообщения зависит от параметров, установленных в диалоговом окне Excel Безопасность (доступ к которому можно получить с помощью команды Сервис⇨Макрос⇨Безопасность). В этом окне опция Доверять доступ к Visual Basic Project по умолчанию не выставлена. Даже если пользователь разрешит запускать макросы, которые содержатся в рабочей книге, но этот параметр будет отключен, макрос не сможет внести изменения в проект VBA. Обратите внимание, что данный параметр применяется ко всем рабочим книгам и не может изменяться только для одной рабочей книги.



Прямой доступ к этому параметру получить невозможно. Единственный способ определить наличие в среде этого параметра — попытаться получить доступ к объекту `VBProject`, после чего проверить существование ошибки. Следующий код иллюстрирует этот способ.

```
On Error Resume Next
Set x = ActiveWorkbook.VBProject
If Err <> 0 Then
  MsgBox "Настройки безопасности не позволяют выполнить макрос"
  Exit Sub
End If
```

Не все примеры, приведенные в этой главе, предназначены для использования конечными пользователями. Большинство из них призваны упростить задачу разработчиков по созданию новых проектов. В таких проектах может возникнуть необходимость в отключении параметра Доверять доступ к Visual Basic Project.

ПЕРЕКРЕСТНАЯ ССЫЛКА

Дополнительная информация об автоматизации OLE приведена в главе 20.

После получения предоставления о работе объектной модели IDE можно приступать к созданию кода, который будет выполнять целый ряд полезных функций.

- Создавать и удалять модули VBA.
- Вставлять код VBA.
- Создавать пользовательские диалоговые окна.
- Добавлять элементы управления в диалоговые окна `UserForm`.

Объектная модель IDE

Программирование IDE требует четкого понимания объектной модели. Объектом верхнего уровня в иерархии объектов выступает VBE (Visual Basic Environment — среда разработки Visual Basic). Как и в случае с объектной моделью Excel, VBE содержит другие объекты. Упрощенная версия объектной иерархии IDE выглядит следующим образом.

```
VBE
  VBProject
    VBComponent
      CodeModule
          Designer
```

```
        Property
    Reference
  Window
  CommandBar
```

Эта глава не содержит описания коллекций `Windows` и `CommandBars`, предоставленных в Extensibility Library, так как они практически бесполезны для разработчиков приложений Excel. Основное внимание будет уделено объекту `VBProject`, который часто используется разработчиками (читайте об этом во врезке "Важные замечания для пользователей Excel 2003").

Коллекция VBProjects

Каждая открытая рабочая книга или надстройка представлена объектом `VBProject`. Для того чтобы получить доступ к объекту `VBProject`, представляющему рабочую книгу, необходимо воспользоваться свойством `VBProject` объекта `Workbook`. Следующий оператор создает переменную, которая представляет объект `VBProject` активной рабочей книги.

```
Dim VBP As VBProject
Set VBP = ActiveWorkbook.VBProject
```

Если при выполнении оператора `Dim` выводится сообщение об ошибке, то необходимо проверить существование ссылки на библиотеку Microsoft Visual Basic for Application Extensibility.

Каждый объект `VBProject` содержит коллекцию компонентов VBA, которые входят в проект (диалоговые окна `UserForm`, модули кода, модули классов, а также модули документов). Данная коллекция называется `VBComponents`. Кроме того, объект `VBProject` содержит коллекцию `References` текущего проекта, представляющую библиотеки, на которые ссылается этот проект.

Не существует возможности непосредственно добавить новый элемент в коллекцию `VBProjects`. Данная задача выполняется в результате открытия или создания рабочей книги в Excel. Новый элемент автоматически будет добавлен в коллекцию `VBProjects`. Подобно этому, невозможно непосредственно удалить объект `VBProject`. Чтобы решить такую задачу, необходимо закрыть рабочую книгу.

Коллекция VBComponents

Для того чтобы получить доступ к члену коллекции `VBComponents`, необходимо воспользоваться свойством `VBComponents` с указанием индексного номера или имени. Следующие операторы демонстрируют два способа получения доступа к компонентам VBA.

```
Set VBC = ThisWorkbook.VBProject.VBComponents(1)
Set VBC = ThisWorkbook.VBProject.VBComponents("Module1")
```

Коллекция References

Каждый проект VBA в Excel содержит определенное количество ссылок. Ссылки проекта можно добавлять, удалять и редактировать с помощью команды Tools⇨References (Сервис⇨Ссылки) (рис. 28.1). Каждый проект содержит ссылки на библиотеки объектов (например, VBA, Excel, OLE Automation, а также Office Object Library), а при необходимости в проект можно добавить новые ссылки.

Ссылками в проекте можно также управлять с помощью кода VBA. Коллекция `References` содержит объекты `Reference`, класс которых предоставляет все необходимые свойства и методы. Приведенная далее процедура отображает окно сообщения, которое содержит значения свойств `Name`, `Description` и `FullPath` каждого объекта `Reference` проекта активной рабочей книги.

```
Sub ListReferences()
    Dim Ref As Reference
    Msg = ""
    For Each Ref In ActiveWorkbook.VBProject.References
        Msg = Msg & Ref.Name & vbCrLf
        Msg = Msg & Ref.Description & vbCrLf
        Msg = Msg & Ref.FullPath & vbCrLf & vbCrLf
    Next Ref
    MsgBox Msg
End Sub
```

На рис. 28.2 показан результат выполнения этой процедуры. Активная рабочая книга содержит пять ссылок.

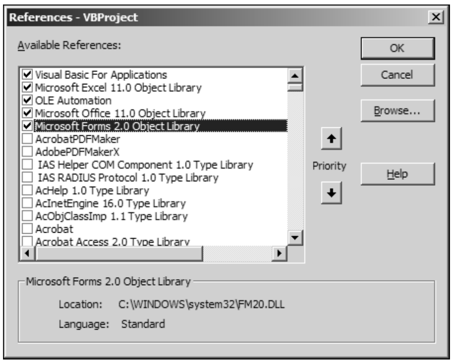


Рис. 28.1. Диалоговое окно References *(Ссылки) отображает ссылки, которые присутствуют в проекте*

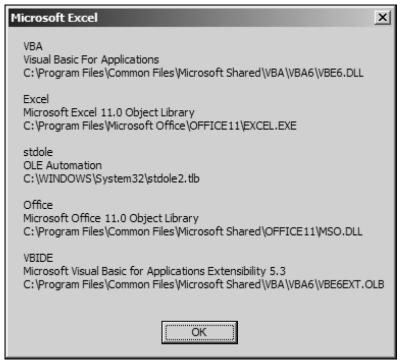


Рис. 28.2. Данное окно сообщения отображает информацию о ссылках проекта активной рабочей книги

По причине того, что переменная объекта имеет тип `Reference`, процедура `ListReferences` требует наличия ссылки на VBA Extensibility Library. Если объявить переменную `Ref` как имеющую универсальный тип `Object`, то ссылка на VBA Extensibility Library не понадобится.

Ссылки можно добавлять программно. Для этого используется один из двух методов класса `Reference`. Метод `AddFromFile` добавляет ссылку, если известно имя файла и путь к папке, в которой он хранится. Метод `AddFromGuid` добавляет ссылку, если известно значение *глобально уникального идентификатора,* или GUID. Дополнительная информация по этой теме приводится в справочном руководстве.

Первый пример

Процедура `ShowComponents`, показанная в листинге 28.1, просматривает каждый компонент VBA в активной рабочей книге и записывает следующую информацию на рабочий лист:

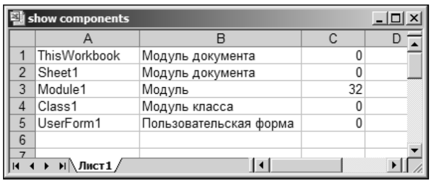
- название компонента;
- тип компонента;
- количество строк кода в модуле кода этого компонента.

Листинг 28.1. Отображение на рабочем листе всех активных компонентов VBA

```
Sub ShowComponents()
    Dim VBP As VBProject
    Set VBP = ActiveWorkbook.VBProject
    NumComponents = VBP.VBComponents.Count
    Cells.ClearContents
    For i = 1 To NumComponents
'       Название
        Cells(i, 1) = VBP.VBComponents(i).Name

'       Тип
        Select Case VBP.VBComponents(i).Type
            Case 1
                Cells(i, 2) = "Модуль"
            Case 2
                Cells(i, 2) = "Модуль класса"
            Case 3
                Cells(i, 2) = "Пользовательская форма"
            Case 100
                Cells(i, 2) = "Модуль документа"
        End Select
'   Количество строк кода
        Cells(i, 3) = _
         VBP.VBComponents(i).CodeModule.CountOfLines
    Next i
End Sub
```

На рис. 28.3 показан результат выполнения процедуры `ShowComponents`. В этом случае проект VBA содержит пять компонентов, и только один из них имеет модуль кода.



	A	B	C	D
1	ThisWorkbook	Модуль документа	0	
2	Sheet1	Модуль документа	0	
3	Module1	Модуль	32	
4	Class1	Модуль класса	0	
5	UserForm1	Пользовательская форма	0	
6				

Рис. 28.3. Результат выполнения процедуры `ShowComponents`

Эта рабочая книга доступна на прилагаемом к книге компакт-диске. Обратите внимание на то, что рабочая книга содержит ссылку на VBA Extensibility Library.

Замещение модуля обновленной версией

Пример, приведенный в этом разделе, демонстрирует замещение модуля VBA другим модулем VBA. Кроме демонстрации трех методов объекта `VBComponent` (`Export`, `Remove` и `Import`), процедура имеет и практическое применение. Например, после того, как рабочая книга была распространена среди группы пользователей, выяснилось, что в макросе допущена ошибка, поэтому он требует обновления. Так как пользователи могли добавить данные в рабочую книгу, ее полное замещение может оказаться нецелесообразным. Решением в такой ситуации служит распространение другой рабочей книги, которая содержит макрос, замещающий модуль VBA обновленной версией, хранящейся в файле.

Приводимый пример состоит из двух рабочих книг.

- `UserBook.xls` — эта рабочая книга содержит модуль (`Module1`), который необходимо заместить.
- `UpdateUserBook.xls` — содержит процедуру VBA, замещающую `Module1` в рабочей книге `UserBook.xls` на обновленную версию.

Процедура `BeginUpdate`, показанная в листинге 28.2, находится в рабочей книге `UpdateUserBook`, которая распространяется среди пользователей рабочей книги `UserBook.xls`. Эта процедура позволяет удостовериться, что рабочая книга `UserBook.xls` открыта. Пользователю выдается сообщение о необходимости обновления модуля (рис. 28.4).

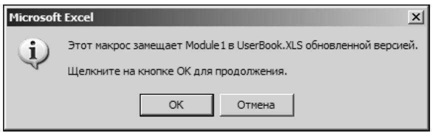


Рис. 28.4. Данное окно сообщения предупреждает пользователя об обновлении модуля

Листинг 28.2. Предоставление пользователю информации о замене модуля

```
Sub UpdateUserBook()
    Filename = "UserBook.xls"

'   Активизация рабочей книги
    On Error Resume Next
    Workbooks(Filename).Activate
    If Err <> 0 Then
        MsgBox Filename & " необходимо открыть!", vbCritical
        Exit Sub
    End If

    Msg = "Этот макрос замещает Module1 в UserBook.XLS "
    Msg = Msg & "обновленной версией" & vbCrLf & vbCrLf
    Msg = Msg & "Щелкните на кнопке OK для продолжения"
    If MsgBox(Msg, vbInformation + vbOKCancel) = vbOK Then
        Call ReplaceModule
    Else
        MsgBox "Модуль не заменен!", vbCritical
    End If
End Sub
```

Щелкнув на кнопке OK, вы вызовете процедуру `ReplaceModule`. Эта процедура, показанная в листинге 28.3, заменяет существующий модуль `Module1`, который находится в рабочей книге `UserBook.xls`, на обновленную версию из рабочей книги `UpdateUserBook`.

Листинг 28.3. Обновление существующего модуля кода

```
Sub ReplaceModule()
'   Экспорт Module1 из текущей книги
    Filename = ThisWorkbook.Path & "\tempmodxxx.bas"
    ThisWorkbook.VBProject.VBComponents("Module1") _
      .Export Filename

'   Замена Module1 в файле UserBook.xls
    Set VBP = ActiveWorkbook.VBProject
    On Error GoTo ErrHandle
```

```
    With VBP.VBComponents
        .Remove VBP.VBComponents("Module1")
        .Import Filename
    End With

'   Удаление временного файла
    Kill Filename
    MsgBox "Модуль успешно заменен", vbInformation
    Exit Sub

ErrHandle:
'   Ошибка?
        MsgBox "ОШИБКА. Невозможно заменить модуль", _
          vbCritical
End Sub
```

Такая процедура делится на следующие этапы.

1. Сначала в файл экспортируется модуль `Module1` (обновленная версия). Файлу присваивается случайное имя, чтобы снизить вероятность перезаписи существующего файла.
2. Из рабочей книги `UserForm.xls` удаляется модуль `Module1` (обновляемая версия). Для этого используется метод `Remove` коллекции `VBComponents`.
3. После этого процедура импортирует модуль (сохраненный на первом этапе) в рабочую книгу `UserBook.xls`.
4. Удаляется временный файл с обновленной версией модуля.
5. Выводится окно с сообщением, которое содержит информацию о результате обновления. Обработка ошибок необходима для своевременного извещения пользователя о возникновении ошибки.

Этот пример (использующий два файла) доступен на прилагаемом к книге компакт-диске. По причине того, что макрос записывает информацию в текущую папку, перед его запуском копию примера необходимо сохранить на жестком диске.

Использование VBA для создания кода VBA

Пример, приведенный в этом разделе, демонстрирует использование VBA для создания дополнительного кода VBA. Процедура `AddSheetAndButton` выполняет следующие действия.

1. Добавляет новый рабочий лист.
2. На рабочий лист добавляет элемент управления `CommandButton`.
3. Меняет расположение, размер и название элемента управления `CommandButton`.
4. Затем добавляет процедуру обработки события для элемента управления `CommandButton`. Такая процедура называется `CommandButton1_Click`. Она располагается в модуле кода рабочего листа и используется для активизации рабочего листа `Лист1`.

В листинге 28.4 приведен исходный код процедуры `AddSheetAndButton`.

Листинг 28.4. Генерация нового рабочего листа, встроенной кнопки и процедуры обработки события

```
Sub AddSheetAndButton()
    Dim NewSheet As Worksheet
    Dim NewButton As OLEObject

'   Добавление листа
    Set NewSheet = Sheets.Add

'   Добавление кнопки
    Set NewButton = NewSheet.OLEObjects.Add _
      ("Forms.CommandButton.1")
    With NewButton
        .Left = 4
        .Top = 4
        .Width = 100
        .Height = 24
        .Object.Caption = "Вернуться к Лист1"
    End With

'   Добавление обработчика событий
    Code = "Sub CommandButton1_Click()" & vbCrLf
    Code = Code & "    On Error Resume Next" & vbCrLf
    Code = Code & "    Sheets(""Лист1"").Activate" & vbCrLf
    Code = Code & "    If Err <> 0 Then" & vbCrLf
    Code = Code & "      MsgBox ""Невозможно активизировать Лист1.""" & vbCrLf
    Code = Code & "    End If" & vbCrLf
    Code = Code & "End Sub"

    With ActiveWorkbook.VBProject. _
      VBComponents(NewSheet.Name).CodeModule
        NextLine = .CountOfLines + 1
        .InsertLines NextLine, Code
    End With
End Sub
```

На рис. 28.5 показана рабочая книга, которая была получена в результате выполнения процедуры `AddSheetAndButton`.

Самой сложной задачей процедуры является добавление кода VBA в модуль кода нового рабочего листа. Код хранится в переменной, которая называется `Code`. Каждый оператор отделен последовательностью символов возврата каретки и перевода строки. Метод `InsertLines` добавляет код в модуль кода добавленного рабочего листа.

Переменная `NextLine` хранит количество существующих строк в модуле кода и каждый раз при добавлении строки увеличивает свое значение на единицу. Таким образом обеспечивается вставка кода в конец модуля. Если вставлять код, начиная с первой строки, а система пользователя настроена на автоматическое добавление оператора `Option Explicit` в модуль, то возникнет сообщение об ошибке.

На рис. 28.6 показана процедура, которая создана в результате выполнения процедуры `AddSheetAndButton`.

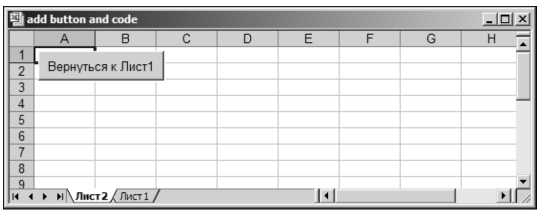


Рис. 28.5. Этот рабочий лист, элемент управления `CommandButton`, а также процедура обработки события добавлены кодом VBA

Добавление элементов управления в диалоговое окно UserForm на этапе разработки

Тот, кто потратил не одну бессонную ночь на создание диалоговых окон `UserForm`, знает, что данная задача может быть утомительной, поскольку после добавления элемента управления необходимо изменить его размер и расположение, чтобы правильно выровнять относительно остальных элементов управления в диалоговом окне. Даже если полностью использовать потенциал команд форматирования VBE, настройка внешнего вида элементов управления может занять немало времени.

Диалоговое окно `UserForm`, которое показано на рис. 28.7, содержит 100 элементов управления `CommandButton`. Все они имеют одинаковый размер и выровнены в диалоговом окне. Кроме того, каждый элемент управления `CommandButton` имеет собственную процедуру обработки события. Добавление элементов управления вручную и создание для них процедур обработки событий может занять довольно много времени — точнее, почти бесконечность. Автоматическое добавление этих элементов управления на этапе разработки с помощью кода VBA займет всего несколько секунд.

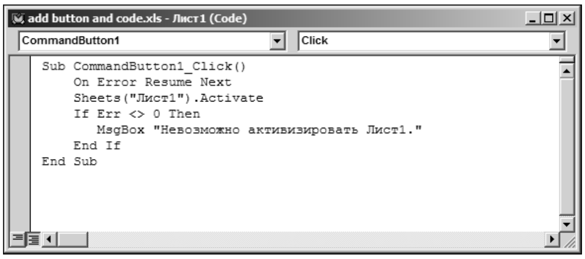


Рис. 28.6. Эта процедура обработки события создана с помощью кода VBA

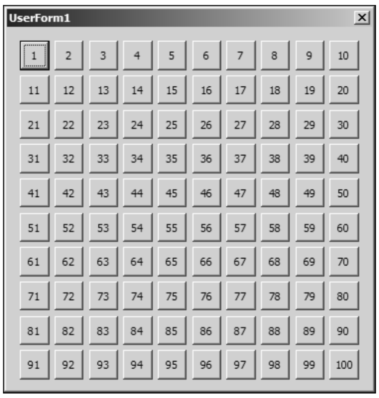


Рис. 28.7. Для добавления элементов управления `CommandButton` в диалоговое окно `UserForm` используется процедура VBA

Управление диалоговыми окнами UserForm на этапе разработки и на этапе выполнения

Важно понимать разницу между управлением диалоговым окном `UserForm` и его элементами управления на этапе разработки и на этапе выполнения. Управление диалоговым окном `UserForm` на этапе выполнения становится заметно для пользователя, поскольку диалоговое окно отображается на экране. Но изменения, внесенные на этапе выполнения, непостоянные. Если создать код, который на этапе выполнения будет изменять значение свойства `Caption` диалогового окна, то после отображения этого диалогового окна пользователь увидит новый заголовок. Если же после этого вернуться в редактор VBE, то окажется, что диалоговое окно `UserForm` имеет исходный заголовок. Часть IV этой книги содержит ряд примеров, которые демонстрируют методы управления диалоговыми окнами `UserForm` и их элементами управления на этапе выполнения.

С другой стороны, управление на этапе разработки имеет постоянную природу. Оно является аналогом ручного изменения свойств элементов управления и диалоговых окон с помощью инструментов редактора VBE. Как правило, код VBA на этапе разработки используется для автоматизации операций над создаваемыми элементами управления и диалоговыми окнами `UserForm`. Для того чтобы изменить элементы управления на этапе разработки, необходимо получить доступ к объекту `Designer` диалогового окна `UserForm`.

Чтобы продемонстрировать разницу между управлением на этапе разработки и управлением на этапе выполнения, были созданы две простые процедуры, которые добавляют элемент управления `CommandButton` в диалоговое окно `UserForm`. Одна процедура добавляет элемент управления на этапе выполнения, а вторая — на этапе разработки.

Процедура `RunTimeButton`, приведенная ниже, очень проста. Она находится в модуле кода общего назначения и предназначена для создания элемента управления `CommandButton` и изменения значений его свойств. После добавления элемента управления отображается диалоговое окно `UserForm`. Элемент управления `CommandButton` можно увидеть в диалоговом окне в момент его появления на экране. Если просмотреть диалоговое окно в редакторе VBE, то окажется, что элемент управления на форме отсутствует.

```
Sub RunTimeButton()
'   Добавление кнопки на этапе выполнения
    Dim Butn As CommandButton
    Set Butn = UserForm1.Controls.Add("Forms.CommandButton.1")
    With Butn
        .Caption = "На этапе выполнения"
        .Width = 100
        .Top = 10
    End With
    UserForm1.Show
End Sub
```

Ниже приведена процедура `DesignTimeButton`. Отличие заключается в том, что она использует объект `Designer`, который содержится в объекте `VBComponent`. В частности, применяется метод `Add` для добавления элемента управления `CommandButton`. Поскольку процедура обращается к объекту `Designer`, элемент управления `CommandButton` добавляется в диалоговое окно `UserForm` так же, как это выполняется при ручном редактировании диалогового окна.

```
Sub DesignTimeButton()
'   Добавление кнопки на этапе разработки
    Dim Butn As CommandButton
    Set Butn = ThisWorkbook.VBProject. _
      VBComponents("UserForm1") _
      .Designer.Controls.Add("Forms.CommandButton.1")
    With Butn
        .Caption = "На этапе разработки"
        .Width = 120
        .Top = 40
    End With
End Sub
```

Добавление 100 элементов управления CommandButton на этапе разработки

Пример данного раздела демонстрирует преимущества использования объекта `Designer` при создании диалоговых окон `UserForm`. В этом случае код добавляет 100 элементов управления `CommandButton` (которые соответствующим образом расположены и выровнены), устанавливает значение свойства `Caption` для каждого элемента управления `CommandButton` и создает 100 процедур обработки событий (для каждого элемента управления `CommandButton`).

Листинг 28.5 приводит полный исходный код процедуры `Add100Buttons`.

Листинг 28.5. Генерация диалогового окна UserForm, содержащего 100 кнопок

```
Sub Add100Buttons()
  Dim UFvbc As Object 'VBComponent
  Dim CMod As Object 'Модуль кода
  Dim ctl As Control
  Dim cb As CommandButton
  Dim n As Integer, c As Integer, r As Integer
  Dim code As String

  Set UFvbc = ThisWorkbook.VBProject.VBComponents("UserForm1")

' Удаление всех элементов управления
  For Each ctl In UFvbc.Designer.Controls
    UFvbc.Designer.Controls.Remove ctl.Name
  Next ctl

' Удаление всего кода VBA
  UFvbc.CodeModule.DeleteLines 1, UFvbc.CodeModule.CountOfLines

' Добавление 100 элементов управления CommandButton
  n = 1
  For r = 1 To 10
    For c = 1 To 10
      Set cb = UFvbc.Designer.Controls.Add("Forms.CommandButton.1")
      With cb
        .Width = 22
        .Height = 22
        .Left = (c * 26) - 16
        .Top = (r * 26) - 16
        .Caption = n
      End With

'     Добавление кода процедуры обработки события
      With UFvbc.CodeModule
        code = ""
        code = code & "Private Sub CommandButton" & n & "_Click" & vbCr
```

```
        code = code & "Msgbox ""This is CommandButton" & n & """" & vbCr
        code = code & "End Sub"
        .InsertLines .CountOfLines + 1, code
      End With
      n = n + 1
    Next c
  Next r
  VBA.UserForms.Add("UserForm1").Show
End Sub
```

Процедура Add100Buttons требует наличия диалогового окна, которое называется UserForm1. Процедура начинает работу с удаления всех элементов управления на этом диалоговом окне. Затем удаляется весь код в модуле кода данного диалогового окна. Для этого используется метод DeleteLines объекта CodeModule. Далее добавляются элементы управления CommandButton и процедуры обработки события для этих элементов управления с помощью двух циклов For-Next. Процедуры обработки событий очень просты. Ниже приведен пример одной из них (процедура задана для элемента управления CommandButton1).

```
Private Sub CommandButton1_Click()
  MsgBox "Это CommandButton1"
End Sub
```

Если при работе процедуры очищается диалоговое окно и его модуль кода, то довольно просто тестировать значения расстояния между кнопками, а также значения размера самих кнопок. Достаточно изменить несколько значений и перезапустить процедуру, чтобы получить представление о результате. При этом удалять созданные элементы управления перед повторным запуском процедуры не требуется.

Если необходимо отобразить диалоговое окно после добавления элементов управления на этапе разработки, то рекомендуется добавить следующий оператор непосредственно перед оператором End Sub.

```
  VBA.UserForms.Add("UserForm1").Show
```

Вам потребуется немало времени, чтобы выяснить, как необходимо отображать диалоговое окно UserForm. Если интерпретатор VBA генерирует диалоговое окно UserForm, в котором насчитывается 100 кнопок, то это диалоговое окно содержится в памяти интерпретатора, но оно не является частью проекта. Для того чтобы формально добавить диалоговое окно UserForm1 в коллекцию UserForms, необходимо воспользоваться методом Add. Возвращаемое этим методом значение (да, этот метод возвращает значение) является ссылкой на диалоговое окно. Именно поэтому в конце вызова метода Add необходимо вызвать метод Show. Из этого следует правило, согласно которому для отображения созданного диалогового окна его сначала необходимо добавить в коллекцию UserForms.

Программное создание диалоговых окон UserForm

В завершение данной главы речь пойдет об использовании средств VBA для создания диалоговых окон UserForm на этапе выполнения. В этом разделе представлены два примера: один из них довольно прост, второй — намного сложнее.

Простой пример

Пример, приведенный в этом разделе, не имеет особой практической ценности. Точнее, он совершенно бесполезен. Однако этот пример демонстрирует важные концепции. Процедура `MakeForm` выполняет следующие задачи.

1. Создает временное диалоговое окно `UserForm` в активной рабочей книге. Для этого используется метод `Add` коллекции `VBComponents`.
2. Добавляет элемент управления `CommandButton` в диалоговое окно `UserForm`. Для этого применяется объект `Designer`.
3. Добавляет процедуру обработки события в модуль кода диалогового окна `UserForm`. Данная процедура (`CommandButton1_Click`) используется для отображения окна сообщения и закрытия диалогового окна.
4. Отображает диалоговое окно `UserForm`.
5. Удаляет диалоговое окно `UserForm`.

Общим результатом работы процедуры является динамическое создание диалогового окна `UserForm`, его использование и удаление после использования. Приводимый пример, а также пример в следующем разделе стирают границу между модификацией диалоговых окон на этапе разработки и на этапе выполнения. Диалоговое окно создается с помощью методов разработки, но все это происходит на этапе выполнения.

> Документация по созданию диалоговых окон `UserForm` предоставляет довольно мало информации, касающейся рассматриваемой темы. При разработке этой процедуры основным является метод проб и ошибок.

Ниже приведен полный листинг процедуры `MakeForm`.

Листинг 28.6. Динамическое создание диалогового окна UserForm

```
Sub MakeForm()
    Dim TempForm  As Object 'VBComponent
    Dim NewButton As Msforms.CommandButton
    Dim Line As Integer
    Dim TheForm

    Application.VBE.MainWindow.Visible = False

'   Создание диалогового окна UserForm
    Set TempForm = ThisWorkbook.VBProject. _
      VBComponents.Add(3) 'vbext_ct_MSForm
    With TempForm
        .Properties("Caption") = "Временная форма"
        .Properties("Width") = 200
        .Properties("Height") = 100
    End With

'   Добавление элемента управления CommandButton
    Set NewButton = TempForm.Designer.Controls _
      .Add("forms.CommandButton.1")
    With NewButton
        .Caption = "Щелкни!"
        .Left = 60
        .Top = 40
    End With
```

```
'   Добавление процедуры обработки события для CommandButton
    With TempForm.CodeModule
        Line = .CountOfLines
        .InsertLines Line + 1, "Sub CommandButton1_Click()"
        .InsertLines Line + 2, "MsgBox ""Привет!"""
        .InsertLines Line + 3, "Unload Me"
        .InsertLines Line + 4, "End Sub"
    End With

'   Отображение диалогового окна UserForm
    VBA.UserForms.Add(TempForm.Name).Show
'
'   Удаление диалогового окна UserForm
    ThisWorkbook.VBProject.VBComponents.Remove TempForm
End Sub
```

Эта процедура создает диалоговое окно, показанное на рис. 28.8.

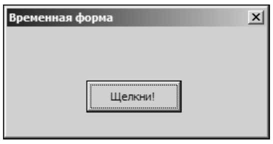


Рис. 28.8. Такое диалоговое окно `UserForm`, а также связанный с ним код, созданы динамически с помощью кода VBA



Рабочая книга, которая содержит процедуру `MakeForm`, не требует создания ссылки на VBA Extensibility Library, так как переменная `TempForm` объявляется как имеющая универсальный тип `Object` (а не объект `VBComponent`). Более того, процедура не использует встроенные константы.

Обратите внимание на то, что первая инструкция процедуры скрывает окно VBE, устанавливая его свойство `Visible` в значение `False`. Это позволяет избежать мерцания экрана при генерации кода и диалогового окна.

Сложный пример

Данный пример имеет как образовательную, так и практическую ценность. Он состоит из функции `GetOption`, которая отображает диалоговое окно `UserForm`. В этом диалоговом окне находится несколько элементов управления `OptionButton`, подписи которых указываются в виде аргументов функции. Функция возвращает значение, которое соответствует элементу управления `OptionButton`, выбранному пользователем. Полный исходный код функции приводится в листинге 28.7.

Листинг 28.7. Динамическая генерация диалогового окна с несколькими переключателями

```
Function GetOption(OpArray, Default, Title)
    Dim TempForm As Object 'VBComponent
    Dim NewOptionButton As Msforms.OptionButton
    Dim NewCommandButton1 As Msforms.CommandButton
    Dim NewCommandButton2 As Msforms.CommandButton
    Dim i As Integer, TopPos As Integer
    Dim MaxWidth As Long
    Dim Code As String

'   Скрытие окна VBE для избежания мерцания
    Application.VBE.MainWindow.Visible = False

'   Создание диалогового окна UserForm
    Set TempForm = _
```

```
      ThisWorkbook.VBProject.VBComponents.Add(3) 'vbext_ct_MSForm
    TempForm.Properties("Width") = 800

'   Добавление элементов управления OptionButton
    TopPos = 4
    MaxWidth = 0
    For i = LBound(OpArray) To UBound(OpArray)
        Set NewOptionButton = TempForm.Designer.Controls. _
          Add("forms.OptionButton.1")
        With NewOptionButton
            .Width = 800
            .Caption = OpArray(i)
            .Height = 15
            .Left = 8
            .Top = TopPos
            .Tag = i
            .AutoSize = True
            If Default = i Then .Value = True
            If .Width > MaxWidth Then MaxWidth = .Width
        End With
        TopPos = TopPos + 15
    Next i

'   Добавление кнопки Отмена
    Set NewCommandButton1 = TempForm.Designer.Controls. _
      Add("forms.CommandButton.1")
    With NewCommandButton1
        .Caption = "Отмена"
        .Height = 18
        .Width = 44
        .Left = MaxWidth + 12
        .Top = 6
    End With

'   Добавление кнопки OK
    Set NewCommandButton2 = TempForm.Designer.Controls. _
      Add("forms.CommandButton.1")
    With NewCommandButton2
        .Caption = "OK"
        .Height = 18
        .Width = 44
        .Left = MaxWidth + 12
        .Top = 28
    End With

'   Добавление процедур обработки событий для CommandButton

    Code = ""
    Code = Code & "Sub CommandButton1_Click()" & vbCrLf
    Code = Code & "  GETOPTION_RET_VAL=False" & vbCrLf
    Code = Code & "  Unload Me" & vbCrLf
    Code = Code & "End Sub" & vbCrLf
    Code = Code & "Sub CommandButton2_Click()" & vbCrLf
    Code = Code & "  Dim ctl" & vbCrLf
    Code = Code & "  GETOPTION_RET_VAL = False" & vbCrLf
    Code = Code & "  For Each ctl In Me.Controls" & vbCrLf
    Code = Code & "    If TypeName(ctl) = ""OptionButton"" Then" & vbCrLf
    Code = Code & "      If ctl Then GETOPTION_RET_VAL = ctl.Tag" & vbCrLf
    Code = Code & "    End If" & vbCrLf
    Code = Code & "  Next ctl" & vbCrLf
    Code = Code & "  Unload Me" & vbCrLf
    Code = Code & "End Sub"

    With TempForm.CodeModule
        .InsertLines .CountOfLines + 1, Code
    End With
```

```
'   Изменение параметров диалогового окна
    With TempForm
        .Properties("Caption") = Title
        .Properties("Width") = NewCommandButton1.Left + _
           NewCommandButton1.Width + 10
        If .Properties("Width") < 160 Then
            .Properties("Width") = 160
            NewCommandButton1.Left = 106
            NewCommandButton2.Left = 106
        End If
        .Properties("Height") = TopPos + 24
    End With

'   Отображение диалогового окна
    VBA.UserForms.Add(TempForm.Name).Show

'   Удаление диалогового окна
    ThisWorkbook.VBProject.VBComponents.Remove VBComponent:=TempForm

'   Передача выбранного параметра в вызывающую процедуру
    GetOption = GETOPTION_RET_VAL
End Function
```

Функция GetOption работает достаточно быстро, учитывая количество действий, которое выполняется в фоновом режиме. Во многих компьютерах диалоговое окно отображается практически мгновенно. После выполнения своей функции диалоговое окно UserForm удаляется.

ИСПОЛЬЗОВАНИЕ ФУНКЦИИ GETOPTION

Функция GetOption принимает три аргумента.

- *OpArray* — строковый массив, содержащий названия элементов, которые будут отображены в диалоговом окне с помощью элементов управления OptionButton.
- *Default* — целое число, которое указывает на элемент управления OptionButton, выбранный по умолчанию в момент отображения диалогового окна UserForm. Если этот параметр имеет значение 0, то не выбран ни один из элементов управления OptionButton.
- *Title* — текст, который будет отображаться в строке заголовка созданного диалогового окна UserForm.

ПРИНЦИП ДЕЙСТВИЯ ФУНКЦИИ GETOPTION

Функция GetOption выполняет следующие операции.

1. Скрывает окно VBE, что позволяет избежать мерцания экрана в момент генерации диалогового окна UserForm и добавления кода процедур обработки событий.
2. Создает диалоговое окно UserForm и назначает его переменной TempForm.
3. Затем в диалоговое окно добавляются элементы управления OptionButton. Для этого используется массив, который передан в функцию с помощью аргумента OpArray. Свойство Tag элемента управления используется для сохранения индексного номера. Значение свойства Tag выбранного элемента управления возвращается функцией в качестве результата.
4. После этого функция добавляет два элемента управления CommandButton: кнопку OK и кнопку Отмена.

5. Для каждого из элементов управления `CommandButton` создается процедура обработки события.
6. Проводятся завершающие действия по настройке диалогового окна `UserForm`: функция изменяет расположение кнопок `CommandButton` и размер диалогового окна `UserForm`.
7. Диалоговое окно `UserForm` отображается на экране. Когда пользователь щелкает на кнопке OK, выполняется процедура `CommandButton1_Click`. Она определяет, какой из элементов управления `OptionButton` выделен, и присваивает соответствующее значение переменной `GETOPTION_RET_VAL` (эта переменная объявлена с областью действия `Public`).
8. Диалоговое окно `UserForm` закрывается и удаляется.
9. Функция возвращает значение переменной `GETOPTION_RET_VAL` в качестве результата.



Заметным преимуществом динамического создания диалогового окна `UserForm` является размещение функции в единственном модуле и отсутствие необходимости в создании ссылки на VBA Extensibility Library. Таким образом, полученный модуль можно просто экспортировать (модуль называется `modOptionsForm`). Затем вы вправе импортировать модуль в рабочую книгу и получить доступ к функции `GetOption`.

Следующая процедура демонстрирует использование функции `GetOption`. В этом случае диалоговое окно `UserForm` предоставляет пять вариантов выбора (которые содержатся в массиве `Ops`).

```
Sub TestGetOption()
    Dim Ops(1 To 5)
    Dim UserOption

    On Error Resume Next
    Dim x
    Set x = ActiveWorkbook.VBProject
    If Err <> 0 Then
        MsgBox "Настройки безопасности не позволяют выполнить макрос", vbCriti-
cal
        On Error GoTo 0
        Exit Sub
    End If

    Ops(1) = "Север"
    Ops(2) = "Юг"
    Ops(3) = "Запад"
    Ops(4) = "Восток"
    Ops(5) = "Все регионы"
    UserOption = GetOption(Ops, 5, "Выберите регион")
    MsgBox Ops(UserOption)
End Sub
```

Переменная `UserOption` хранит индексный номер выбранного переключателя. Если пользователь щелкнет на кнопке Отмена, то переменная `UserOption` будет установлена в значение `False`.

На рис. 28.9 показано диалоговое окно `UserForm`, которое создается такой функцией.

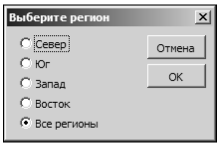


Рис. 28.9. Это диалоговое окно `UserForm` создано функцией `GetOption`

Диалоговое окно UserForm меняет собственный размер для того, чтобы подстроиться под заданное количество элементов управления, добавленных в диалоговое окно. Теоретически, функция UserOption может принимать массив любого размера. Однако необходимо ограничить количество элементов массива, так как размер диалогового окна UserForm не может превысить размер экрана.

ЗАДАЧИ, ВЫПОЛНЯЕМЫЕ ФУНКЦИЕЙ GETOPTION

Ниже приведены процедуры обработки события для элементов управления CommandButton. Этот код генерируется с помощью функции GetOption и размещается в модуле кода созданного диалогового окна UserForm.

```
Sub CommandButton1_Click()
  GETOPTION_RET_VAL = False
  Unload Me
End Sub

Sub CommandButton2_Click()
  Dim ctl
  GETOPTION_RET_VAL = False
  For Each ctl In Me.Controls
    If TypeName(ctl) = "OptionButton" Then
      If ctl Then GETOPTION_RET_VAL = ctl.Tag
    End If
  Next ctl
  Unload Me
End Sub
```

По причине того, что диалоговое окно UserForm после использования уничтожается, его внешний вид невозможно оценить с помощью редактора VBE. Для того чтобы рассмотреть диалоговое окно UserForm в редакторе, превратите следующий оператор в комментарий, указав перед ним символ апострофа ('):

```
ThisWorkbook.VBProject.VBComponents.Remove _
VBComponent:=TempForm
```

Глава 29

Принципы управления модулями классов

В ЭТОЙ ГЛАВЕ...

В данной главе описываются основные методы использования модулей классов, а также приводятся примеры, которые помогут понять принципы управления модулями классов.

- Введение в модули классов
- Применение модулей классов
- Примеры, демонстрирующие ключевые концепции использования модулей классов

Для большинства программистов на VBA концепция модуля класса — это неизведанная тайна. Модули классов поддерживаются в Excel 97 и более поздних версиях программы.

Что такое модуль класса

Модуль класса (class module) — это специальный тип модуля VBA, который можно добавить в проект. Проще говоря, модуль класса позволяет разработчику создать новый класс объектов. На этом этапе вы уже должны знать, что программирование Excel сводится к управлению объектами. Модуль класса позволяет создать новые объекты, а также соответствующие свойства, методы и события.



Примеры из предыдущих глав этой книги демонстрируют использование модулей классов (см. главы 15, 18, 19 и 23).

У вас может возникнуть вопрос: "Нужно ли создавать новые объекты?". Ответ краток: нет. Однако как только вы поймете преимущества модулей классов, то можете рискнуть и попробовать. В большинстве случаев модуль класса служит достойной заменой функциям и процедурам, но можно найти для него и более серьезное применение. Иногда модуль класса является единственным способом получения необходимого результата.

Ниже приводится список типичных вариантов использования модулей классов.

- *Обработка событий, связанных со встроенными диаграммами.* (Соответствующий пример представлен в главе 18).
- *Контроль над событиями уровня приложения.* В число таких событий входит активизация рабочей книги. (Соответствующие примеры находятся в главах 19 и 23.)

- *Перенастройка функций Windows API для упрощения их использования в исходном коде.* Например, можно создать класс, который упрощает определение или изменение состояния клавиш <NumLock> и <CapsLock>. Также можно создать класс, который упрощает доступ к системному реестру.
- *Выполнение одной процедуры несколькими объектами диалогового окна* `UserForm`. Обычно объект имеет собственную процедуру обработки события. Пример в главе 15 демонстрирует использование модуля класса, предназначенного для предоставления нескольким элементам управления `CommandButton` одной процедуры обработки события `Click`.
- *Создание компонентов, рассчитанных на повторное использование; их можно импортировать в другие проекты.* Как только будет создан модуль класса общего назначения, его можно импортировать в другие проекты для сокращения объема работ по разработке приложений.

Пример: создание класса NumLock

В данном разделе приводится пошаговая инструкция создания полезного, хотя и достаточно простого, модуля класса. Этот модуль класса создает класс `NumLock`, который имеет одно свойство: `Value`. Определение или установка значения клавиши <NumLock> требует использования нескольких функций Windows API. Такой модуль класса предназначен для упрощения операции управления клавишей <NumLock>. Вызовы функций API, а также необходимый код расположены в модуле класса (а не в обычном модуле кода VBA). В чем преимущества такого подхода? Работать с подобным кодом намного проще, а модуль класса можно использовать в других проектах.

Создав класс в коде VBA, определите текущее состояние клавиши <NumLock> с помощью инструкции, отображающей значение свойства `Value`.

```
MsgBox NumLock.Value
```

Также в коде можно изменить состояние клавиши <NumLock>. Например, следующий оператор включает клавишу <NumLock>.

```
NumLock.Value = True
```

Кроме того, с помощью кода вы можете переключать режим NumLock следующим образом.

```
NumLock.Toggle
```

Важно понимать, что модуль класса содержит код, который определяет объект, включая его свойства и методы. В результате предоставляется возможность создания экземпляров этого класса в модуле кода VBA общего назначения, а также использования его свойств и методов.

Для того чтобы лучше понять процесс создания модуля класса, воспользуйтесь инструкциями следующего раздела. Начнем с создания новой рабочей книги.

Вставка модуля класса

Запустите редактор VBE и выберите команду Insert⇨Class Module (Вставка⇨Модуль класса). Это приведет к добавлению пустого модуля класса, который называется `Class1`. Если окно Properties не отображено на экране, нажмите клавишу <F4>. После отображения окна Properties измените имя модуля класса на `NumLock` (рис. 29.1).

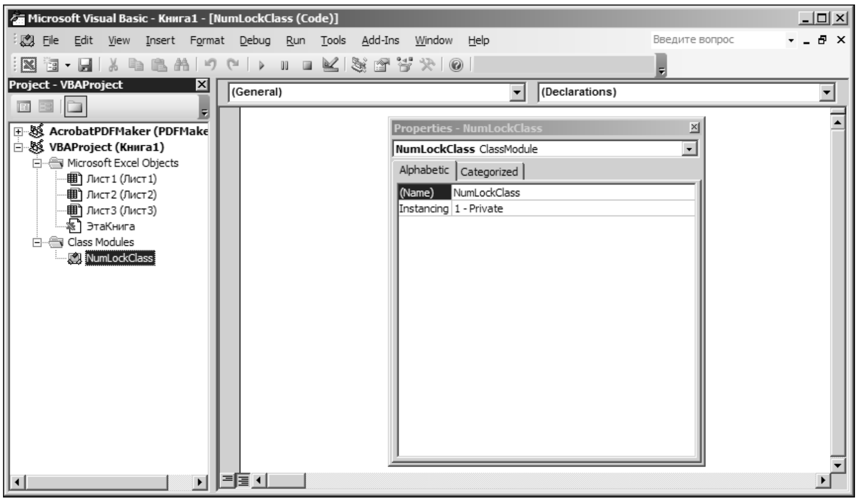


Рис. 29.1. Пустой модуль класса, который называется `NumLockClass`

Добавление кода VBA

Далее необходимо создать код, обеспечивающий работоспособность свойства `Value`. Чтобы определить или изменить состояние клавиши <NumLock>, модуль класса должен включать объявления функций Windows API, которые используются для установки или получения состояния клавиши <NumLock>. В листинге 29.1 приведен код объявления данных.

Листинг 29.1. Объявление данных

```
' Объявление типов
Private Type OSVERSIONINFO
    dwOSVersionInfoSize As Long
    dwMajorVersion As Long
    dwMinorVersion As Long
    dwBuildNumber As Long
    dwPlatformId As Long
    szCSDVersion As String * 128
End Type

' Объявление функций API
Private Declare Function GetVersionEx Lib "Kernel32" _
    Alias "GetVersionExA" _
    (lpVersionInformation As OSVERSIONINFO) As Long

Private Declare Sub keybd_event Lib "user32" _
    (ByVal bVk As Byte, _
    ByVal bScan As Byte, _
    ByVal dwflags As Long, ByVal dwExtraInfo As Long)

Private Declare Function GetKeyboardState Lib "user32" _
    (pbKeyState As Byte) As Long

Private Declare Function SetKeyboardState Lib "user32" _
    (lppbKeyState As Byte) As Long

'Объявление констант
Const VK_NUMLOCK = &H90
```

```
Const VK_SCROLL = &H91
Const VK_CAPITAL = &H14
Const KEYEVENTF_EXTENDEDKEY = &H1
Const KEYEVENTF_KEYUP = &H2
Const VER_PLATFORM_WIN32_NT = 2
Const VER_PLATFORM_WIN32_WINDOWS = 1
```

После этого создайте процедуру, которая будет получать текущее состояние клавиши <NumLock>. Она определяет свойств `Value` создаваемого объекта. В качестве имени свойства можно использовать любое название. `Value` — неплохой вариант. Для того чтобы получить текущее состояние клавиши, вставьте в модуль следующую процедуру `Property Get`.

```
Property Get Value() As Boolean
'   Получение текущего состояния
    Dim keys(0 To 255) As Byte
    GetKeyboardState keys(0)
    Value = keys(VK_SCROLL)
End Property
```



Подробно код процедур группы `Property` рассматривается далее в этой главе. Для получения необходимой информации обратитесь к разделу "Программирование свойств".

Процедура, использующая функцию Windows API `GetKeyboardState`, определяет текущее состояние клавиши <NumLock>. Эта процедура вызывается каждый раз, когда код VBA запрашивает значение свойства `Value`. Например, оператор VBA, который представлен ниже, приводит к выполнению процедуры `Property Get`.

```
MsgBox NumLock.Value
```

Теперь необходимо создать процедуру, которая будет устанавливать состояние клавиши <NumLock>. Клавиша может иметь одно из двух состояний: включена или выключена. Такая задача выполняется с помощью следующей процедуры `Property Let` (листинг 29.2).

Листинг 29.2. Процедура `Property Let`

```
Property Let Value(boolVal As Boolean)
    Dim o As OSVERSIONINFO
    Dim keys(0 To 255) As Byte
    o.dwOSVersionInfoSize = Len(o)
    GetVersionEx o
    GetKeyboardState keys(0)
'   Проверка состояния
    If boolVal = True And keys(VK_SCROLL) = 1 Then Exit Property
    If boolVal = False And keys(VK_SCROLL) = 0 Then Exit Property
'   Переключение режима
    If o.dwPlatformId = VER_PLATFORM_WIN32_WINDOWS Then '(Win95)
        'Переключение функции клавиши
        keys(VK_SCROLL) = IIf(keys(VK_SCROLL) = 0, 1, 0)
        SetKeyboardState keys(0)
    ElseIf o.dwPlatformId = VER_PLATFORM_WIN32_NT Then ' (WinNT)
        'Эмуляция нажатия клавиши
        keybd_event VK_SCROLL, &H45, KEYEVENTF_EXTENDEDKEY Or 0, 0
        'Эмуляция нажатия клавиши
        keybd_event VK_SCROLL, &H45, KEYEVENTF_EXTENDEDKEY Or KEYEVENTF_KEYUP, 0
    End If
End Property
```

Процедура `Property Let` принимает один аргумент, который может иметь либо значение `True`, либо `False`. Оператор VBA, который приводится ниже, устанавливает значение `Value` объекта `NumLock` в значение `True`. Для этого вызывается процедура `Property Let`.

```
NumLock.Value = True
```

Наконец, вам потребуется процедура переключения режима NumLock (листинг 29.3).

Листинг 29.3. Процедура `Toggle`

```
Sub Toggle()
'   Переключение состояния
    Dim o As OSVERSIONINFO
    o.dwOSVersionInfoSize = Len(o)
    GetVersionEx o
    Dim keys(0 To 255) As Byte
    GetKeyboardState keys(0)
    If o.dwPlatformId = VER_PLATFORM_WIN32_WINDOWS Then '(Win95)
        'Переключение режима
        keys(VK_SCROLL) = IIf(keys(VK_SCROLL) = 0, 1, 0)
        SetKeyboardState keys(0)
    ElseIf o.dwPlatformId = VER_PLATFORM_WIN32_NT Then ' (WinNT)
        'Эмуляция нажатия клавиши
        keybd_event VK_SCROLL, &H45, KEYEVENTF_EXTENDEDKEY Or 0, 0
        'Эмуляция отпускания клавиши
        keybd_event VK_SCROLL, &H45, KEYEVENTF_EXTENDEDKEY Or KEYEVENTF_KEYUP, 0
    End If
End Sub
```

Обратите внимание, что в последнем листинге приведена стандартная процедура. Тем не менее, она полностью справляется с возложенной на нее задачей.

Использование класса NumLock

Перед применением модуля класса `NumLockClass` необходимо создать экземпляр класса. Оператор, который находится в модуле кода VBA общего назначения (не в модуле класса), создает такой экземпляр.

```
Dim NumLock As New NumLockClass
```

Обратите внимание на то, что тип объекта указан как `NumLockClass` (т.е. используется название модуля класса). Сама переменная может иметь любое название, но `NumLock` является наиболее приемлемым.

Представленная далее процедура устанавливает свойство `Value` объекта `NumLock` в значение `True`. Это приводит к включению клавиши <NumLock>.

```
Sub NumLockOn()
    Dim NumLock As New NumLockClass
    NumLock.Value = True
End Sub
```

Представленная ниже процедура отображает окно сообщения, которое указывает на текущее состояние клавиши <NumLock> (`True` — если клавиша включена, и `False` — в противном случае).

```
Sub GetNumLockState()
    Dim NumLock As New NumLockClass
    MsgBox NumLock.Value
End Sub
```

Следующая процедура переключает состояние клавиши <NumLock>.

```
Sub ToggleNumLock()
    Dim NumLock As New NumLockClass
    NumLock.Toggle
End Sub
```

Наконец, последняя процедура переключает состояние клавиши <NumLock> без использования метода `Toggle`.

```
Sub ToggleNumLock2()
    Dim NumLock As New NumLockClass
    NumLock.Value = Not NumLock.Value
End Sub
```

Завершенный модуль класса этого примера находится на прилагаемом к книге компакт-диске. Рабочая книга также содержит модуль класса для определения и установки состояния клавиш <CapsLock> и <ScrollLock>.

Еще о модулях классов

Пример предыдущего раздела продемонстрировал методы создания нового объекта, обладающего свойством `Value`. Но объект может иметь любое количество свойств. Кроме того, объект поддерживает методы и события.

Именование класса объектов

Имя, которое используется для модуля класса, является одновременно и именем объекта класса. По умолчанию модули классов получают такие имена, как `Class1`, `Class2` и т.д. Модулю класса требуется более точное имя.

Программирование свойств

Многие объекты обладают как минимум одним свойством. Однако объект может иметь любое количество свойств. После определения свойства его используют в коде, для чего применяется стандартный синтаксис с разделителем-точкой.

объект.`свойство`

Параметр редактора VBE Auto List Members применяется и к объектам, определенным в модуле класса. Это упрощает определение свойств и методов при написании кода.

Свойства объекта, определяемого в модуле класса, могут предназначаться только для чтения, только для записи, а также для чтения и записи. Свойство, предназначенное только для чтения, определяется с помощью ключевого слова `Property Get`. Ниже приведен пример процедуры с использованием оператора `Property Get`.

```
Property Get FileNameOnly() As String
    FileNameOnly = ""
    For i = Len(FullName) To 1 Step -1
        Char = Mid(FullName, i, 1)
        If Char = "\" Then
            Exit Function
        Else
            FileNameOnly = Char & FileNameOnly
        End If
    Next i
End Property
```

Несложно заметить, что процедура `Property Get` работает как функция. Код рассчитывает и возвращает значение свойства, которое соответствует названию процедуры. В приведенном примере в качестве имени процедуры используется `FileNameOnly`. Процедура возвращает имя файла, которое содержится в значении общедоступной переменной `FullName`. Например, если переменная `FullName` содержит значение `c:\windows\myfile.txt`, то процедура возвращает значение `myfile.txt`. Процедура `FileNameOnly` вызывается тогда, когда код VBA обращается к свойству объекта.

Для свойств, ориентированных на чтение и запись, необходимо создать две процедуры: `Property Get` (отвечает за чтение значения свойства) и `Property Let` (устанавливает значение свойства). Значение, определенное свойству, рассматривается в качестве последнего (или единственного) аргумента процедуры `Property Get`.

Ниже приводятся примеры подобных процедур.

```
Property Get SaveAsExcelFile() As Boolean
    SaveAsExcelFile = XLFile
End Property

Property Let SaveAsExcelFile(boolVal As Boolean)
   XLFile = boolVal
End Property
```

Если свойство имеет объектный тип данных, вместо `Property Let` необходимо использовать `Property Set`.

Переменная, объявленная в модуле класса с областью действия `Public`, может также применяться в качестве свойства объекта. В представленном выше примере избавимся от процедур `Property Get` и `Property Let`, а также заменим их на объявление переменной на уровне модуля.

```
Public SaveAsExcelFile As Boolean
```

Если необходимо активизировать свойство, предназначенное только для записи, воспользуйтесь процедурой `Property Let` без соответствующей процедуры `Property Get`.

Предыдущие примеры предполагают существование булевой переменной `XLFile`, которая определена на уровне модуля. Процедура `Property Get` возвращает значение этой переменной в качестве значения свойства. Если объект называется `FileSys`, то следующий оператор будет отображать текущее значение свойства `SaveAsExcelFile`.

```
MsgBox FileSys.SaveAsExcelFile
```

С другой стороны, процедура `Property Let` принимает аргумент и использует значение аргумента для изменения значения свойства. Например, можно создать оператор (приведенный ниже), который будет устанавливать свойство `SaveAsExcelFile` в значение `True`.

```
FileSys.SaveAsExcelFile = True
```

В данном случае значение `True` передается процедуре `Property Let`, которая и изменяет значение свойства.

В предыдущих примерах используется переменная уровня модуля — `XLFile`. Именно эта переменная используется для хранения значения свойства. Такую переменную необходимо создавать для каждого свойства, которое объявляется в модуле класса.

Процедуры свойств подчиняются всем правилам именования процедур. Может оказаться, что VBA не позволит использовать в названиях процедур отдельные зарезервированные слова. Если при создании процедуры свойства выводится извещение о синтаксической ошибке, то рекомендуем изменить название процедуры.

Методы программирования

Метод объекта класса программируется подобно другим процедурам. Для этого используются ключевые слова Sub или Function, которые размещаются в модуле класса. Как правило, объект имеет методы, однако может существовать и без них. Для вызова метода в коде используются стандартные соглашения.

```
объект.метод
```

Как и любой другой метод VBA, метод объекта класса выполняет определенные действия. Следующая процедура представляет метод, который сохраняет рабочую книгу в одном из двух форматов. Используемый формат определяется значением свойства XLFile. Легко заметить, что в этой процедуре нет ничего необычного.

```
Sub SaveFile()
    If XLFile Then
        ActiveWorkbook.SaveAs FileName:=FName, _
          FileFormat:=xlWorkbookNormal
    Else
        ActiveWorkbook.SaveAs FileName:=FName, _
          FileFormat:=xlCSV
    End If
End Sub
```

Пример, приведенный в следующем разделе, существенно прояснит концепцию свойств и методов объекта, определенного в модуле класса.

События модуля класса

Каждый модуль класса поддерживает два события: Initialize и Terminate. Событие Initialize возникает при создании нового экземпляра класса, а событие Terminate — при уничтожении объекта. Событие Initialize можно использовать для установки значений свойств, принятых по умолчанию.

Шаблон для процедур обработки событий выглядит следующим образом.

```
Private Sub Class_Initialize()
'    Здесь располагается код инициализации
End Sub

Private Sub Class_Terminate()
'    Здесь располагается код уничтожения
End Sub
```

Объект *уничтожается* (и освобождается выделенная для него память), когда процедура или модуль, в котором он объявлен, завершает свое выполнение. Объект можно уничтожить в любой момент, присвоив ему значение Nothing. Следующий оператор уничтожает объект, который называется MyObject.

```
Set MyObject = Nothing
```

Пример: класс CSVFile

Пример, приведенный в этом разделе, определяет такой класс объектов, как CSVFileClass. Этот класс имеет два свойства и два метода.

СВОЙСТВА

- ExportRange — предназначено для чтения и записи. Указывает диапазон ячеек рабочего листа, который будет экспортирован в файл формата CSV.
- ImportRange — предназначено для чтения и записи. Указывает диапазон, в который будет импортирована информация из файла формата CSV.

МЕТОДЫ

- `Import` — используется для импортирования информации из файла в формате CSV, который определяется значением аргумента `CSVFileName`. Информация импортируется в диапазон, обозначенный свойством `ImportRange`.
- `Export` — используется для экспортирования диапазона, определенного значением свойства `ExportRange`, в файл формата CSV, который указывается значением аргумента `CSVFileName`.

Переменные уровня модуля класса

Модуль класса должен поддерживать собственные локальные переменные, которые отражают значения свойств объекта. Модуль класса `CSVFileClass` содержит две переменные, которые используются для хранения значений свойств. Такие переменные объявляются в начале модуля.

```
Private RangeToExport As Range
Private ImportToCell As Range
```

`RangeToExport` — это объект `Range`, который представляет экспортируемый диапазон. `ImportToCell` является объектом `Range`, представляющим верхнюю левую ячейку того диапазона, в который будут импортироваться данные из файла в формате CSV. Эти переменные получают значения с помощью процедур `Property Get` и `Property Let` (см. следующий раздел).

Процедуры свойств

Процедуры свойств модуля класса `CSVFileClass` показаны в листинге 29.4. Процедуры `Property Get` используются для получения значения переменной, а процедуры `Property Let` — для установки значения переменной.

Листинг 29.4. Процедуры свойств модуля CSVFileClass

```
Property Get ExportRange() As Range
    Set ExportRange = RangeToExport
End Property

Property Let ExportRange(rng As Range)
    Set RangeToExport = rng
End Property

Property Get ImportRange() As Range
    Set ImportRange = ImportToCell
End Property

Property Let ImportRange(rng As Range)
    Set ImportToCell = rng
End Property
```

Процедуры методов

Модуль класса `CSVFileClass` содержит две процедуры, которые представляют методы класса. Эти процедуры рассматриваются в следующем разделе.

ПРОЦЕДУРА EXPORT

Процедура `Export`, приведенная в листинге 29.5, вызывается каждый раз при обращении к методу `Export`. Процедура принимает один аргумент: полное имя файла, который будет хранить экспортируемый диапазон. Процедура обеспечивает базо-

вый уровень обработки ошибок. Например, проверяется наличие значения у свойства ExportRange. Для этого проверяется значение переменной RangeToExport. Процедура создает обработчик для возникающих ошибок.

Листинг 29.5. Экспорт диапазона ячеек рабочего листа с помощью метода модуля класса

```
Sub Export(CSVFileName)
'   Экспортирует диапазон в файл CSV
    Dim ExpBook As Workbook

    If RangeToExport Is Nothing Then
        MsgBox "Экспортируемый диапазон не определен"
        Exit Sub
    End If

    On Error GoTo ErrHandle
    Application.ScreenUpdating = False
    Set ExpBook = Workbooks.Add(xlWorksheet)
    RangeToExport.Copy
    Application.DisplayAlerts = False
    With ExpBook
        .Sheets(1).Paste
        .SaveAs FileName:=CSVFileName, FileFormat:=xlCSV
        .Close SaveChanges:=False
    End With
    Application.CutCopyMode = False
    Application.ScreenUpdating = True
    Application.DisplayAlerts = True
    Exit Sub
ErrHandle:
    ExpBook.Close SaveChanges:=False
    Application.CutCopyMode = False
    Application.ScreenUpdating = True
    Application.DisplayAlerts = True
    MsgBox "ОШИБКА " & Err & vbCrLf & vbCrLf & Error(Err), _
      vbCritical, "Ошибка метода экспортирования"
End Sub
```

Процедура Export при выполнении копирует указанный с помощью переменной RangeToExport диапазон во временную рабочую книгу. После этого рабочая книга сохраняется в файле формата CSV, и файл закрывается. Так как обновление экрана не выполняется, пользователь не видит происходящего. Если возникнет ошибка (например, будет указано неверное имя файла), процедура перейдет к метке ErrHandle и отобразит окно сообщения, которое содержит номер ошибки и ее описание.

ПРОЦЕДУРА IMPORT

Процедура Import, приведенная в листинге 29.6, импортирует данные из файла CSV, который указан с помощью аргумента CSVFileName. Содержимое файла копируется в диапазон, указанный с помощью значения переменной ImportToCell. Эта переменная используется для хранения значения свойства ImportRange. В данном случае обновление экрана не выполняется — пользователь не должен иметь возможности наблюдать за работой процедуры. Как и Export, процедура Import содержит базовые функции обработки ошибок.

Листинг 29.6. Импортирование содержимого файла в указанный диапазон с помощью метода модуля класса

```
Sub Import(CSVFileName)
'   Импортирует файл CSV в диапазон
    Dim CSVFile As Workbook

    If ImportToCell Is Nothing Then
        MsgBox "Диапазон для импорта не определен"
        Exit Sub
    End If

    If CSVFileName = "" Then
        MsgBox "Не определено имя импортируемого файла"
        Exit Sub
    End If

    On Error GoTo ErrHandle
    Application.ScreenUpdating = False
    Application.DisplayAlerts = False
    Workbooks.Open CSVFileName
    Set CSVFile = ActiveWorkbook
    ActiveSheet.UsedRange.Copy Destination:=ImportToCell
    CSVFile.Close SaveChanges:=False
    Application.ScreenUpdating = True
    Application.DisplayAlerts = True
    Exit Sub
ErrHandle:
    CSVFile.Close SaveChanges:=False
    Application.ScreenUpdating = True
    Application.DisplayAlerts = True
    MsgBox "ОШИБКА " & Err & vbCrLf & vbCrLf & Error(Err), _
      vbCritical, "Ошибка метода импортирования"
End Sub
```

Использование объекта CSVFileClass

Для того чтобы создать экземпляр класса `CSVFileClass` в собственном коде VBA, необходимо создать переменную, которая имеет тип `CSVFileClass`. Ниже приведен пример объявления такой переменной.

```
Dim CSVFile As New CSVFileClass
```

Иногда может потребоваться объявить переменную указанного типа, а также создать объект в тот момент, когда он будет запрашиваться в коде. Для этого используются операторы `Dim` и `Set`.

```
Dim CSVFile As CSVFileClass
' Здесь может находится другой код
Set CSVFile = New CSVFile
```

Преимуществом применения операторов `Dim` и `Set` является то, что объект не создается до тех пор, пока в коде не встретится соответствующий оператор `Set`. При этом очевидна экономия памяти: если объект не нужен, память под него не выделяется. Например, код может следовать алгоритму, который точно определяет, когда необходимо создавать объект. Кроме того, использование оператора `Set` позволяет создавать несколько экземпляров одного класса.

После создания экземпляра объекта допускается вводить другие операторы, которые будут обращаться к свойствам и методам созданного объекта, определенного в модуле класса.

Функция редактора VBE Auto List Members выполняется для всех объектов. После ввода имени переменной с точкой в конце будет отображен список свойств и методов, которые определены для этого объекта.

Следующая процедура демонстрирует сохранение текущего выделенного диапазона ячеек в файл CSV, который называется `temp.csv`. Этот файл находится в той же папке, что и рабочая книга.

```
Sub ExportARange()
    Dim CSVFile As New CSVFileClass
    With CSVFile
        .ExportRange = ActiveWindow.RangeSelection
        .Export CSVFileName:=ThisWorkbook.Path & "\temp.csv"
    End With
End Sub
```

Использовать структуру `With-End With` не обязательно. Например, процедура может выглядеть следующим образом.

```
Sub ExportARange()
    Dim CSVFile As New CSVFileClass
    CSVFile.ExportRange = ActiveWindow.RangeSelection
    CSVFile.Export CSVFileName:=ThisWorkbook.Path & "\temp.csv"
End Sub
```

Представленная далее процедура демонстрирует операцию импортирования файла CSV. Данные размещаются, начиная с активной ячейки.

```
Sub ImportAFile()
    Dim CSVFile As New CSVFileClass
    With CSVFile
    On Error Resume Next
        .ImportRange = ActiveCell
        .Import CSVFileName:=ThisWorkbook.Path & "\temp.csv"
    End With
    If Err <> 0 Then _
      MsgBox "Невозможно импортировать " & ThisWorkbook.Path & "\temp.csv"
End Sub
```

VBA может работать с большим количеством экземпляров класса. Следующий код создает массив из трех объектов `CSVFileClass`.

```
Sub Export3Files()
    Dim CSVFile(1 To 3) As New CSVFileClass
    CSVFile(1).ExportRange = Range("A1:A20")
    CSVFile(2).ExportRange = Range("B1:B20")
    CSVFile(3).ExportRange = Range("C1:C20")

    For i = 1 To 3
        CSVFile(i).Export CSVFileName:="File" & i & ".csv"
    Next i
End Sub
```

Глава 30

Часто задаваемые вопросы о программировании в Excel

В ЭТОЙ ГЛАВЕ...

Те пользователи, которые часто “блуждают” в Internet, знают, что такое FAQ (ЧаВо). Это список *часто задаваемых вопросов (frequently asked questions)* (и ответов на них), касающихся определенной темы. Такие списки вопросов, в основном, публикуются в конференциях, целью которых является сокращение потока одинаковых вопросов на определенную тему. Но зачастую такие документы не выполняют возложенную на них функцию, так как одинаковые вопросы продолжают повторяться.

- Список часто задаваемых вопросов о программировании в Excel
- Где получить ответ на вопрос, который в книге отсутствует?

Как только я понял, что люди в большинстве своем задают одинаковые вопросы о программировании на VBA, то создал данный документ, который содержит ответы на часто задаваемые вопросы о программировании в Excel 97 и более поздних версиях. Представляемый документ не ответит на все возникающие вопросы, однако он поможет решить ряд распространенных задач и даст представление о принципах решения некоторых проблем. Вопросы (и многие ответы) были взяты из следующих источников.

- Группы новостей `microsoft.public.excel.*`.
- Группы новостей `comp.apps.spreadsheet`.
- База знаний Microsoft.
- Почтовые сообщения, отправленные на мое имя.

Если данная глава не содержит ответа на ваш вопрос...

В таком случае обратитесь к предметному указателю этой книги. В книге приведено много информации, которую вряд ли можно назвать ответом на часто задаваемые вопросы. Если окажется, что и в книге не содержится ответ на интересующий вас вопрос, то обратитесь к источникам, указанным в приложении А.

Все задаваемые вопросы были объединены в несколько категорий. Каждый вопрос относится к одной из следующих тем.

- Общие вопросы об Excel.
- Вопросы о редакторе Visual Basic.

- Вопросы о процедурах и функциях.
- Вопросы об объектах, свойствах, методах и событиях.
- Вопросы о пользовательских формах.
- Вопросы о надстройках.
- Вопросы об объектах `CommandBar`.

В некоторых случаях используется достаточно произвольная классификация, так как вопрос с одинаковым успехом может принадлежать к нескольким категориям. Более того, вопросы в каждой категории излагаются в свободном порядке.

Кстати, бо́льшая часть информации, приведенной в этой главе, подробно рассматривается в других главах настоящей книги.

Общие вопросы об Excel

ПОЧЕМУ В EXCEL ИСПОЛЬЗУЕТСЯ ДВА ЯЗЫКА СОЗДАНИЯ МАКРОСОВ?

Ранние версии Excel использовали макроязык, который назывался XLM. Язык VBA начал использоваться в Excel 5, он намного превосходит по возможностям XLM. Со временем язык XLM был вытеснен из среды разработки, поэтому в данный момент рекомендуется использовать только язык VBA.

МНЕ НЕОБХОДИМО РАСПРОСТРАНЯТЬ РАБОЧУЮ КНИГУ СРЕДИ ПОЛЬЗОВАТЕЛЕЙ EXCEL 4. СУЩЕСТВУЕТ ЛИ СПОСОБ ЗАПИСИ ДЕЙСТВИЙ НА МАКРОЯЗЫКЕ XLM?

Нет. Начиная с Excel 97, команда записи макросов генерирует только код на языке VBA. В основном, не стоит разрабатывать рабочие книги в Excel, версия которой новее, чем версия целевой платформы.

РАБОТАЮТ ЛИ В EXCEL 97 И БОЛЕЕ ПОЗДНИХ ВЕРСИЯХ МАКРОСЫ, СОЗДАННЫЕ С ПОМОЩЬЮ ЯЗЫКА XLM?

В большинстве случае такие макросы выполняются.

МНЕ НЕОБХОДИМА УТИЛИТА, КОТОРАЯ СМОЖЕТ ПРЕОБРАЗОВАТЬ МАКРОСЫ EXCEL 4 В ЯЗЫК VBA. СУЩЕСТВУЕТ ЛИ ТАКАЯ УТИЛИТА?

Нет. Такой утилиты не существует, и очень маловероятно, что она будет создана. Подобные преобразования должны осуществляться вручную. Так как все версии Excel могут выполнять макросы XLM, то не существует причины, по которой было бы необходимо преобразовывать такие макросы. За исключением внедрения новых возможностей, которые предоставляются более поздними версиями Excel.

МОЖНО ЛИ ВЫЗВАТЬ ПРОЦЕДУРУ VBA ИЗ МАКРОСА XLM ДЛЯ EXCEL 4?

Да. Воспользуйтесь функцией XLM `RUN`. Например, следующий макрос применяется для запуска процедуры `Test`, которая находится в модуле `Module1` рабочей книги `Book1.xls`.

```
=RUN(Book1.xls!Module1.Test)
```

СУЩЕСТВУЕТ ЛИ СПОСОБ АВТОМАТИЧЕСКОГО ПРЕОБРАЗОВАНИЯ МАКРОСОВ LOTUS 1-2-3 ИЛИ QUATTRO PRO В МАКРОСЫ НА ЯЗЫКЕ VBA?

Нет. Необходимо переписать эти макросы для Excel вручную.

ГДЕ МОЖНО НАЙТИ ПРИМЕРЫ КОДА VBA?

В Internet находятся тысячи примеров кодов VBA. Хорошей отправной точкой может служить узел автора книги — `http://www.j-walk.com/ss/`

СУЩЕСТВУЕТ ЛИ УТИЛИТА ДЛЯ ПРЕОБРАЗОВАНИЯ ПРИЛОЖЕНИЯ EXCEL В ОТДЕЛЬНЫЙ EXE-ФАЙЛ?

Нет, не существует.

КАК ДОБАВИТЬ В ЯЧЕЙКУ РАСКРЫВАЮЩИЙСЯ СПИСОК, ЧТОБЫ ПОЛЬЗОВАТЕЛЬ МОГ ВЫБИРАТЬ ОДНО ИЗ ДОПУСТИМЫХ ЗНАЧЕНИЙ?

Введите список допустимых значений в один столбец. Если необходимо, столбец можно скрыть. Выберите ячейку или ячейки, которые будут отображать список значений, а затем выполните команду Данные⇨Проверка и перейдите на вкладку Параметры. Из раскрывающегося списка Тип данных выберите Список. В поле Источник укажите адрес диапазона или ссылку на элементы списка, хранящиеся на рабочем листе. Удостоверьтесь, что установлен флажок Список допустимых значений. Если список небольшой, то значения можно просто ввести, разделив их запятой. Такая техника не требует использования макросов.

МОЖНО ЛИ ПРИМЕНЯТЬ МЕТОД С РАСКРЫВАЮЩИМСЯ СПИСКОМ, ЕСЛИ СПИСОК ЗНАЧЕНИЙ НАХОДИТСЯ В ДРУГОМ РАБОЧЕМ ЛИСТЕ ЭТОЙ РАБОЧЕЙ КНИГИ?

Да. Необходимо создать имя для этого списка (например, `ListEntries`). Затем в диалоговом окне Проверка вводимых значений введите **=ListEntries** в поле Источник. Удостоверьтесь, что в поле был введен символ равенства, так как в противном случае метод работать не будет.

Я ИСПОЛЬЗУЮ СВОЙСТВО APPLICATION.CALCULATION ДЛЯ УСТАНОВКИ РУЧНОГО РЕЖИМА ВЫЧИСЛЕНИЙ. НО ЭТОТ ПАРАМЕТР ВЛИЯЕТ НА ВСЕ РАБОЧИЕ КНИГИ, А НЕ ТОЛЬКО НА АКТИВНУЮ.

Свойство `Calculation` принадлежит объекту `Application`. Таким образом, изменение режима относится ко всем открытым рабочим книгам. Невозможно установить режим вычисления для отдельной рабочей книги. В Excel 2000 было включено новое свойство объекта `Worksheet`, которое называется `EnableCalculation`. Когда это свойство установлено в значение `False`, рабочий лист не будет пересчитываться, даже если пользователь явно затребует это действие. Установка данного свойства в значение `True` приводит к пересчету рабочего листа.

КАК УВЕЛИЧИТЬ КОЛИЧЕСТВО СТОЛБЦОВ В РАБОЧЕМ ЛИСТЕ?

Это невозможно. Количество столбцов рабочего листа фиксировано и его нельзя изменить. Microsoft продолжает игнорировать тысячи запросов об увеличении количества столбцов на рабочем листе.

КАК УВЕЛИЧИТЬ КОЛИЧЕСТВО СТРОК РАБОЧЕГО ЛИСТА?

В данном случае ответ такой же, как и на предыдущий вопрос.

СУЩЕСТВУЕТ ЛИ ВОЗМОЖНОСТЬ ИЗМЕНИТЬ ЦВЕТ ЯРЛЫКОВ ЛИСТА?

Если используется Excel 2002 и выше, щелкните правой кнопкой мыши на ярлыке листа и выберите Цвет ярлычка. Предыдущие версии Excel не предоставляют возможности изменения цвета ярлыка.

СУЩЕСТВУЕТ ЛИ ВОЗМОЖНОСТЬ ИЗМЕНИТЬ ШРИФТ ЯРЛЫКОВ ЛИСТА?

Да, но данная задача выполняется за пределами Excel. В Windows откройте папку Панель управления и выберите Экран. В диалоговом окне Свойства: Экран перейдите на вкладку Оформление. В списке Элемент выберите Полоса прокрутки. Воспользуйтесь стрелками для увеличения или уменьшения размера. Изменение этого параметра будет влиять и на другие приложения.

МОЖНО ЛИ ИЗМЕНИТЬ ЦВЕТ И ШРИФТ КОММЕНТАРИЕВ К ЯЧЕЙКАМ, КОТОРЫЙ ПРИНЯТ ПО УМОЛЧАНИЮ?

Да. В Windows в папке Панель управления выберите Экран. В диалоговом окне Свойства: Экран перейдите на вкладку Оформление. В списке Элемент выберите Всплывающая подсказка. Воспользуйтесь предоставленными элементами управления для изменения параметров шрифта. Изменение этого параметра повлияет и на другие приложения.

СУЩЕСТВУЕТ ЛИ ВОЗМОЖНОСТЬ ПРОИГРЫВАТЬ ЗВУКИ В EXCEL?

Да. Вы можете проигрывать файлы в форматах WAV и MIDI, но для этого необходимо воспользоваться функциями Windows API (дополнительная информация приводится в главе 11). Если вы используете Excel 2002 и выше, то обратитесь к новому объекту `Speech`. Представленный далее оператор при выполнении приветствует пользователя по имени (предполагается, что в компьютере установлена звуковая плата).

```
Application.Speech.Speak ("Привет" & Application.UserName)
```

ПРИ КАЖДОМ ЗАПУСКЕ EXCEL ПРОИСХОДИТ СБОЙ. ЧТО С ЭТИМ ДЕЛАТЬ?

При запуске Excel программа автоматически загружает файл `*.xlb`, который содержит настройки панели инструментов. Если этот файл поврежден, то программа не сможет загрузиться. Сбой также происходит при чрезмерно большом размере файла. Обычно размер файла `*.xlb` не должен превышать 500 Кбайт.

Если вы попадаете в ситуацию сбоя Excel при загрузке, то потребуется удалить файл `*.xlb`. Закройте программу. Запустите ее заново. Excel автоматически создаст новый файл настройки панелей инструментов и подключит его.

КАК РАСПЕЧАТАТЬ ПОЛНЫЙ ПУТЬ И ИМЯ ФАЙЛА РАБОЧЕЙ КНИГИ В КОЛОНТИТУЛЕ СТРАНИЦЫ?

Если используется Excel 2002 и выше, то обратитесь к новой возможности диалогового окна Параметры страницы. Если это диалоговое окно отображено, перейдите на вкладку Колонтитулы и щелкните на кнопке Создать колонтитул. Вы увидите новое диалоговое окно, которое используется для добавления кода, обеспечивающего печать пути и имени файла рабочей книги в колонтитуле страницы. Обратите внимание:

если рабочая книга не сохранена, то путь может оказаться неверным (в подобном случае используется путь по умолчанию для сохранения рабочих книг).

Для более старых версий Excel необходимо воспользоваться макросом VBA и обработать событие `WorkbookBeforePrint`. Например, разместите следующую процедуру в модуле кода объекта `ЭтаКнига`. Такой код будет печатать полный путь и имя файла рабочей книги в левой части верхнего колонтитула на каждой странице.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    For Each sht In ThisWorkbook.Sheets
        sht.PageSetup.LeftHeader = ThisWorkbook.FullName
    Next sht
End Sub
```

Редактор Visual Basic

В EXCEL 95 МОИ МОДУЛИ КОДА VBA БЫЛИ РАСПОЛОЖЕНЫ В РАБОЧЕЙ КНИГЕ, НО ЕСЛИ РАБОЧУЮ КНИГУ ОТКРЫТЬ В EXCEL 97, ТО МОДУЛИ КОДА ИСЧЕЗНУТ.

Модули остались в рабочей книге, однако для того, чтобы их просмотреть и отредактировать, необходимо воспользоваться редактором VBE. Комбинация клавиш <Alt+F11> применяется для переключения между Excel и редактором VBE.

МОЖНО ЛИ ИСПОЛЬЗОВАТЬ ПРОГРАММУ ЗАПИСИ МАКРОСОВ VBA ДЛЯ ЗАПИСИ ВСЕХ МАКРОСОВ?

Нет. Запись макросов подходит только для очень простых операций. Макросы, которые задействуют переменные, циклы или прерывают поток выполнения операций, записать невозможно. Но вы вправе воспользоваться командой записи макросов для создания фрагментов кода, а также для получения информации о необходимых свойствах и методах.

EXCEL 95 ПРЕДОСТАВЛЯЛА ВОЗМОЖНОСТЬ УСТАНОВКИ "МЕТКИ" В МАКРОСЕ, ЧТО ПОЗВОЛЯЛО ЗАПИСЫВАТЬ КОМАНДЫ, НАЧИНАЯ С УКАЗАННОГО МЕСТА В СУЩЕСТВУЮЩЕМ МАКРОСЕ. ДОСТУПНА ЛИ ЭТА ВОЗМОЖНОСТЬ В БОЛЕЕ ПОЗДНИХ ВЕРСИЯХ?

Нет. Такая возможность отсутствует, начиная с Excel 97. Для того чтобы добавить код записанного макроса в существующий макрос, необходимо записать макрос, скопировать требуемый код и вставить его в существующий макрос.

Я СОЗДАЛ УНИВЕРСАЛЬНЫЕ МАКРОСЫ, КОТОРЫЕ ДОЛЖНЫ БЫТЬ ДОСТУПНЫ ВСЕ ВРЕМЯ. КАК ОБЕСПЕЧИТЬ ПОСТОЯННЫЙ ДОСТУП К НИМ?

Можно сохранить макросы в персональной книге макросов. Это (обычно) скрытая рабочая книга, которая загружается в Excel автоматически. После записи макроса у вас появляется возможность сохранить его в персональной книге макросов. Ее файл `Personal.xls` находится в папке `\XLStart`.

Я НЕ МОГУ НАЙТИ СВОЮ ПЕРСОНАЛЬНУЮ КНИГУ МАКРОСОВ. ГДЕ ОНА?

Файл `Personal.xls` не будет существовать до тех пор, пока в него не записан хотя бы один макрос.

Я ЗАБЛОКИРОВАЛ ПРОЕКТ VBA С ПОМОЩЬЮ ПАРОЛЯ И ЗАБЫЛ ПАРОЛЬ. СУЩЕСТВУЕТ ЛИ СПОСОБ РАЗБЛОКИРОВАТЬ ПРОЕКТ?

Существует несколько программ взлома паролей, которые предоставляются независимыми производителями. Для их поиска можно воспользоваться одним из поисковых средств в Web. В поиске вам помогут ключевые слова *Excel password*. Существование подобных программ говорит о том, что пароли Excel не настолько надежны, как того хотелось бы.

КАК МОЖНО ЗАПИСАТЬ МАКРОС ДЛЯ ИЗМЕНЕНИЯ ПАРОЛЯ ПРОЕКТА?

Это невозможно. Средства защиты проекта VBA не предоставляются в объектной модели. Скорее всего, такой шаг сделан для усложнения работы программного обеспечения по взлому паролей.

ПРИ ДОБАВЛЕНИИ НОВЫЙ МОДУЛЬ ВСЕГДА НАЧИНАЕТСЯ СО СТРОКИ OPTION EXPLICIT. ЧТО ОНА ОЗНАЧАЕТ?

Если строка `Option Explicit` находится в начале модуля, это означает, что необходимо объявлять все переменные, которые будут использоваться в пределах данного модуля (достаточно неплохая идея). Если требуется, чтобы эта строка не добавлялась в новые модули, запустите VBE и выберите Tools⇨Options (Сервис⇨Параметры). Перейдите на вкладку Editor (Редактор) и сбросьте флажок Require Variable Declaration (Явное описание переменных). После этого можно объявить переменные или переложить задачу определения типа данных на плечи интерпретатора VBA.

ПОЧЕМУ МОЙ КОД VBA ОТОБРАЖАЕТСЯ ДРУГИМ ЦВЕТОМ? МОЖНО ЛИ ИЗМЕНИТЬ ЭТИ ЦВЕТА?

VBA использует цвета для выделения различных текстов в коде: комментариев, ключевых слов, идентификаторов, операторов с синтаксическими ошибками и т.д. Эти цвета можно переопределять, кроме того, допускается изменять шрифты, которые используются для выделения, и т.д. Воспользуйтесь командой Tools⇨Options (и перейдите на вкладку Editor Format (Формат редактора)) в редакторе VBE.

МНЕ НЕОБХОДИМО УДАЛИТЬ МОДУЛЬ VBA С ПОМОЩЬЮ КОДА VBA. ВОЗМОЖНО ЛИ ЭТО?

Да. Следующий код удаляет модуль `Module1` из активной рабочей книги.

```
With ActiveWorkbook.VBProject
    .VBComponents.Remove .VBComponents("Module1")
End With
```

Этот код может не работать в Excel 2002 и более поздних версиях. Дополнительная информация приводится далее.

Я СОЗДАЛ МАКРОС ДЛЯ EXCEL 2000, КОТОРЫЙ ДОБАВЛЯЕТ КОД VBA К ПРОЕКТУ. КОГДА Я ЗАПУСТИЛ ЕГО В EXCEL 2003, ТО ПОЛУЧИЛ СООБЩЕНИЕ ОБ ОШИБКЕ. ЧТО Я ДЕЛАЮ НЕПРАВИЛЬНО?

В Excel 2002 был представлен новый параметр: Доверять доступ к Visual Basic Project. По умолчанию этот параметр отключен. Для того чтобы его включить, выберите команду Сервис⇨Макрос⇨Безопасность и перейдите на вкладку Надежные источники в диалоговом окне Безопасность.

КАК ИЗМЕНИТЬ ПАРАМЕТРЫ БЕЗОПАСНОСТИ МАКРОСОВ ДЛЯ ПОЛЬЗОВАТЕЛЯ? Я ХОЧУ ОТКЛЮЧИТЬ ПОЯВЛЕНИЕ СООБЩЕНИЯ “ЭТА РАБОЧАЯ КНИГА СОДЕРЖИТ МАКРОСЫ”, КОГДА ОТКРЫВАЕТСЯ ПРИЛОЖЕНИЕ.

Изменение уровня безопасности с помощью кода VBA приведет к полной бесполезности всей системы безопасности макросов. Подумайте над этим.

ПРИ ОТКРЫТИИ РАБОЧЕЙ КНИГИ Я ПОЛУЧАЮ СТАНДАРТНОЕ ПРЕДУПРЕЖДЕНИЕ О НАЛИЧИИ МАКРОСОВ. НО Я УДАЛИЛ ВСЕ МАКРОСЫ ИЗ ЭТОЙ РАБОЧЕЙ КНИГИ! ЭТО ВИРУС?

Скорее всего, нет. Кроме удаления макросов, необходимо удалить и модуль VBA, который содержал эти макросы.

Я НЕ ПОНИМАЮ, КАК РАБОТАЕТ ПАРАМЕТР USERINTERFACEONLY ПРИ ЗАЩИТЕ РАБОЧЕГО ЛИСТА.

Для защиты рабочего листа можно использовать следующий оператор.

```
ActiveSheet.Protect UserInterfaceOnly:=True
```

Это приведет к защите листа, но макрос все же сможет вносить изменения в содержимое листа. Важно понимать, что этот параметр не сохраняется вместе с рабочей книгой. Когда рабочая книга будет открыта снова, потребуется повторно выполнить этот оператор, чтобы опять установить защиту `UserInterfaceOnly`.

КАК УЗНАТЬ, НАХОДИТСЯ ЛИ В РАБОЧЕЙ КНИГЕ МАКРОВИРУС?

В редакторе VBE активизируйте проект, который соответствует рабочей книге. Просмотрите все модули кода и обратите внимание на код VBA, который вам не знаком. Обычно код вируса плохо форматирован и содержит большое количество переменных со странными названиями. Еще одной возможностью является использование коммерческих программ для поиска вирусов.

У МЕНЯ ВОЗНИКЛИ ПРОБЛЕМЫ С ОПЕРАТОРОМ КОНКАТЕНАЦИИ (&) В VBA. КАК ТОЛЬКО Я ПЫТАЮСЬ КОНКАТЕНИРОВАТЬ ДВЕ СТРОКИ, ТО ПОЛУЧАЮ СООБЩЕНИЕ ОБ ОШИБКЕ.

Скорее всего, VBA интерпретирует амперсанд как символ объявления типа. Удостоверьтесь, что перед и после оператора конкатенации добавлены пробелы.

НЕ ПОЛУЧАЕТСЯ ЗАСТАВИТЬ РАБОТАТЬ СИМВОЛ ПРОДОЛЖЕНИЯ СТРОКИ В VBA (ПОДЧЕРКИВАНИЕ).

Для продолжения строки используются два символа: пробел и после него символ подчеркивания.

ПОСЛЕ УДАЛЕНИЯ БОЛЬШОГО КОЛИЧЕСТВА КОДА VBA Я ОБРАТИЛ ВНИМАНИЕ НА ТО, ЧТО РАЗМЕР ФАЙЛА XLS НЕ УМЕНЬШИЛСЯ СООТВЕТСТВУЮЩИМ ОБРАЗОМ. ПОЧЕМУ?

Excel не всегда правильно очищает данные. Это иногда приводит к некоторым скрытым проблемам с переменными, которые больше не используются. Одним из способов решения этой проблемы является экспортирование модуля в файл, удаление модуля и его повторное импортирование.

Я РАСПРОСТРАНИЛ ПРИЛОЖЕНИЕ XLS СРЕДИ БОЛЬШОГО КОЛИЧЕСТВА ПОЛЬЗОВАТЕЛЕЙ. В НЕКОТОРЫХ КОМПЬЮТЕРАХ ПРОЦЕДУРЫ ОБРАБОТКИ ОШИБОК НЕ РАБОТАЮТ. ПОЧЕМУ?

Процедуры обработки ошибок не будут работать, если у пользователей установлен параметр Break on All Errors (Остановка при любой ошибке). Этот параметр доступен в диалоговом окне Options (на вкладке General (Общие)) редактора VBE. К сожалению, данный параметр невозможно изменить с помощью кода VBA. Для того чтобы избежать возникновения подобной проблемы, распространяйте приложение в виде надстройки XLA.

Процедуры

КАКАЯ РАЗНИЦА МЕЖДУ ПРОЦЕДУРОЙ VBA И МАКРОСОМ?

Никакой. Термин *макрос (macro)* "пришел" из ранней эпохи развития электронных таблиц. Эти термины в настоящее время являются взаимнозаменяемыми.

ЧТО ТАКОЕ ПРОЦЕДУРА?

Процедура — это набор операторов VBA, который можно вызвать по имени. Если операторы должны возвратить явный результат (например, значение) процедуре, которая их вызывала, то эта процедура называется *функцией* и объявляется как `Function`. В противном случае процедура объявляется как `Sub`.

ЧТО ТАКОЕ ПЕРЕМЕННЫЙ ТИП ДАННЫХ?

Переменные, тип данных которых явно не определен, получают *переменный тип данных*. При использовании таких переменных VBA автоматически меняет тип переменной в соответствии с присвоенным значением. Это особенно полезно при получении значений из рабочего листа, так как заранее не известно, что содержится в ячейках. Рекомендуется явно объявлять тип переменной. Для этого используются операторы `Dim`, `Public` и `Private`. Применение переменных типов данных замедляет работу приложения и вызывает неэффективное использование ресурсов памяти.

КАКАЯ РАЗНИЦА МЕЖДУ МАССИВОМ ПЕРЕМЕННОГО ТИПА И МАССИВОМ ЗНАЧЕНИЙ ПЕРЕМЕННОГО ТИПА?

Ячейка памяти переменного типа может содержать любой тип данных: единственное значение или массив значений (массив переменного типа). Следующий код создает переменную, которая содержит массив.

```
Dim X As Variant
X = Array(30, 40, 50)
```

В нормальном массиве могут находиться элементы определенного типа, включая нетипизированные элементы (имеющие переменный тип). Приведенный ниже оператор создает массив, который состоит из 3-х элементов переменного типа.

```
Dim X (0 To 2) As Variant
```

Несмотря на то, что массив элементов переменного типа и массив, представленный переменной типа `Variant`, концептуально отличаются, способ доступа к элементам массива остается таким же.

ЧТО ТАКОЕ СИМВОЛ ОПРЕДЕЛЕНИЯ ТИПА?

VBA позволяет добавить к имени переменной символ, который будет указывать тип этой переменной. Например, можно объявить переменную MyVar, имеющую тип integer. Для этого к имени переменной добавляется символ %.

```
Dim MyVar%
```

Ниже приведен список символов определения типа, которые поддерживаются в VBA.

- Integer %
- Long &
- Single !
- Double #
- Currency @
- String $

Я НАПИСАЛ ФУНКЦИЮ VBA, КОТОРАЯ ВЫПОЛНЯЕТСЯ ПРИ ВЫЗОВЕ ИЗ ДРУГОЙ ПРОЦЕДУРЫ. НО ЕСЛИ ЕЕ ИСПОЛЬЗОВАТЬ В ФОРМУЛЕ РАБОЧЕГО ЛИСТА, ОНА НЕ РАБОТАЕТ. ЧТО Я ДЕЛАЮ НЕПРАВИЛЬНО?

Функции VBA, которые вызываются в формулах рабочего листа, имеют некоторые ограничения. В общем, они должны быть строго "пассивными" — функция не может вносить изменения в активную ячейку, менять форматирование, открывать рабочие книги или менять активный рабочий лист. Если функция пытается выполнить одно из этих действий, формула возвратит сообщение об ошибке.

Функции могут лишь выполнять вычисления и возвращать результат. Исключением из данного правила является функция VBA MsgBox. Пользовательская функция при каждом вызове может отображать окно сообщения. Такая возможность очень полезна при отладке пользовательских функций.

Я ХОТЕЛ СОЗДАТЬ ПРОЦЕДУРУ, КОТОРАЯ АВТОМАТИЧЕСКИ ЗАДАЕТ ФОРМАТИРОВАНИЕ, ЗАВИСЯЩЕЕ ОТ ВВОДИМЫХ ДАННЫХ. НАПРИМЕР, ЕСЛИ ВВОДИТСЯ ЗНАЧЕНИЕ, БОЛЬШЕЕ 0, ЦВЕТ ФОНА ЯЧЕЙКИ СТАНОВИТСЯ КРАСНЫМ. ВОЗМОЖНО ЛИ ЭТО?

Конечно, и для этого не обязательно программировать. Воспользуйтесь командой Excel Условное форматирование. Доступ к ней можно получить с помощью опции Формат⇨Условное форматирование.

КОМАНДА УСЛОВНОЕ ФОРМАТИРОВАНИЕ ОЧЕНЬ ПОЛЕЗНА, НО ХОТЕЛОСЬ БЫ ВЫПОЛНЯТЬ ПРИ ВВОДЕ ДАННЫХ В ЯЧЕЙКИ И ДРУГИЕ ОПЕРАЦИИ.

Рекомендуем воспользоваться событием Change объекта рабочего листа. Как только будут внесены изменения в какую-либо из ячеек, возникнет событие Change. Если модуль кода объекта Sheet (Лист) содержит процедуру Worksheet_Change, то при возникновении события Change она будет выполняться автоматически.

КАКИЕ ДРУГИЕ ТИПЫ СОБЫТИЙ МОЖНО КОНТРОЛИРОВАТЬ?

Огромное количество! Интерактивное справочное руководство содержит полный список доступных событий.

Я СОЗДАЛ ПРОЦЕДУРУ ОБРАБОТКИ СОБЫТИЯ (SUB WORKBOOK_OPEN), НО ОНА НЕ ЗАПУСКАЕТСЯ ПРИ ОТКРЫТИИ РАБОЧЕЙ КНИГИ. ЧТО Я ДЕЛАЮ НЕПРАВИЛЬНО?

Скорее всего, процедура находится не там, где должна быть. Процедуры обработки событий рабочей книги должны располагаться в модуле кода объекта `ThisWorkbook` (`ЭтаКнига`). Процедуры обработки событий рабочего листа должны находиться в модуле кода объекта `Sheet` (`Лист`), что отображается в окне Project редактора VBE.

Я МОГУ СОЗДАТЬ ПРОЦЕДУРУ ОБРАБОТКИ СОБЫТИЙ ДЛЯ ОПРЕДЕЛЕННОЙ РАБОЧЕЙ КНИГИ. МОЖНО ЛИ СОЗДАТЬ ПРОЦЕДУРУ ОБРАБОТКИ СОБЫТИЙ ДЛЯ ВСЕХ ОТКРЫТЫХ РАБОЧИХ КНИГ?

Да, но для этого необходимо создать модуль класса. Подробная информация приводится в главе 19.

Я МОГУ СОЗДАВАТЬ ФОРМУЛЫ В EXCEL. ИСПОЛЬЗУЮТСЯ ЛИ В VBA ТЕ ЖЕ МАТЕМАТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОПЕРАТОРЫ?

Да. Кроме того, в VBA предоставляются дополнительные операторы, которые нельзя использовать в формулах рабочего листа. Эти операторы перечислены в следующей таблице.

Оператор	Выполняет
`\`	Деление с остатком
`Eqv`	Возвращает значение `True`, если оба аргумента равны `True` или оба аргумента равны `False`
`Imp`	Логическую импликацию двух выражений
`Is`	Сравнение двух объектов
`Like`	Сравнение строк с поддержкой групповых символов
`Xor`	Возвращает `True` лишь в том случае, если один из аргументов равен `True`

КАК ВЫЗВАТЬ ПРОЦЕДУРУ, КОТОРАЯ НАХОДИТСЯ В ДРУГОЙ РАБОЧЕЙ КНИГЕ?

Для этого необходимо воспользоваться методом `Run` объекта `Application`. Представленный далее оператор вызывает процедуру `Macro1`, которая расположена в рабочей книге `Personal.xls`.

```
Run "Personal.xls!Macro1"
```

Кроме того можно создать ссылку на другую рабочую книгу. Для этого воспользуйтесь командой Tools⇨References редактора VBA.

Я ВОСПОЛЬЗОВАЛСЯ VBA ДЛЯ СОЗДАНИЯ НЕСКОЛЬКИХ ФУНКЦИЙ И ХОТЕЛ БЫ ПРИМЕНИТЬ ЭТИ ФУНКЦИИ В КАЧЕСТВЕ ФОРМУЛ РАБОЧЕГО ЛИСТА, НО УКАЗЫВАТЬ ИМЯ РАБОЧЕЙ КНИГИ ПЕРЕД НАЗВАНИЕМ ФУНКЦИИ НЕУДОБНО. СУЩЕСТВУЕТ ЛИ СПОСОБ ОБОЙТИ ЭТУ ПРОБЛЕМУ?

Да. Преобразуйте рабочую книгу, которая содержит определения функций в надстройку Excel. После открытия надстройки определенные в ней функции можно вызывать, не указывая названия рабочей книги.

Кроме того, если вы создадите ссылку на рабочую книгу, которая содержит определенные пользователем функции, то функцию также можно будет использовать без указания названия рабочей книги. Чтобы создать ссылку, необходимо воспользоваться командой Tools⇨References редактора VBE.

Я ХОЧУ, ЧТОБЫ ОПРЕДЕЛЕННАЯ РАБОЧАЯ КНИГА ЗАГРУЖАЛАСЬ КАЖДЫЙ РАЗ, КОГДА ЗАПУСКАЕТСЯ EXCEL. ЖЕЛАТЕЛЬНО ТАКЖЕ, ЧТОБЫ ПРИ ЗАПУСКЕ EXCEL АВТОМАТИЧЕСКИ ВЫПОЛНЯЛСЯ МАКРОС ИЗ ЭТОЙ РАБОЧЕЙ КНИГИ. ВОЗМОЖНО ЛИ ЭТО?

Почему бы и нет? Для того чтобы открыть рабочую книгу автоматически, сохраните ее в папке `\XLStart`. Чтобы автоматически запускать макрос из этой рабочей книги, создайте процедуру `Workbook_Open` в модуле кода объекта `ЭтаКнига`.

У МЕНЯ ЕСТЬ РАБОЧАЯ КНИГА, В КОТОРОЙ НАХОДИТСЯ ПРОЦЕДУРА WORKBOOK_OPEN. СУЩЕСТВУЕТ ЛИ СПОСОБ ПРЕДОТВРАТИТЬ АВТОМАТИЧЕСКИЙ ЗАПУСК ЭТОЙ ПРОЦЕДУРЫ ПРИ ОТКРЫТИИ РАБОЧЕЙ КНИГИ?

Да. Удерживайте клавишу <Shift> при выборе команды Файл⇨Открыть. Для того чтобы предотвратить выполнение процедуры `Workbook_BeforeClose`, удерживайте клавишу <Shift> при закрытии рабочей книги. Использование клавиши <Shift> не позволяет предотвратить запуск этих процедур при открытии надстройки.

МОЖЕТ ЛИ ПРОЦЕДУРА VBA ПОЛУЧИТЬ ДОСТУП К ЗНАЧЕНИЮ ЯЧЕЙКИ В НЕОТКРЫТОЙ РАБОЧЕЙ КНИГЕ?

VBA не в состоянии это сделать, но вы сможете достичь поставленной цели с помощью языка XLM. К счастью, XLM можно вызывать из VBA. Ниже приведен простой пример получения значения ячейки `A1` из листа `Sheet1` рабочей книги `myfile.xls`, которая находится в папке `c:\files`.

```
MsgBox ExecuteExcel4Macro("'c:\files\[myfile.xls]Sheet1'!R1C1")
```

Обратите внимание на то, что ячейка идентифицируется в формате R1C1.

КАК ПРЕДОТВРАТИТЬ ОТОБРАЖЕНИЕ СООБЩЕНИЯ О НЕОБХОДИМОСТИ СОХРАНЕНИЯ РАБОЧЕЙ КНИГИ ПРИ ЗАКРЫТИИ С ПОМОЩЬЮ VBA?

Можно воспользоваться следующим оператором.

```
ActiveWorkbook.Close SaveChanges:=False
```

Вы также вправе установить значение свойства `Saved` объекта `Workbook` равным `True`. Для этого воспользуйтесь таким оператором.

```
ActiveWorkbook.Saved = True
```

При его выполнении файл не сохраняется, поэтому после закрытия рабочей книги все несохраненные изменения будут утеряны.

Более универсальным решением по отмене сообщений в Excel будет использование следующего оператора.

```
Application.DisplayAlerts = False
```

КАК СДЕЛАТЬ ТАК, ЧТОБЫ МАКРОС ЗАПУСКАЛСЯ РЕГУЛЯРНО В ОПРЕДЕЛЕННОЕ ВРЕМЯ?

Для этого необходимо воспользоваться методом `OnTime` объекта `Application`. Это позволяет указать процедуру, которая будет выполняться в определенное время суток. Когда процедура завершает свою работу, необходимо еще раз воспользоваться методом `OnTime`, чтобы запланировать следующий вызов.

КАК СДЕЛАТЬ ТАК, ЧТОБЫ МАКРОС НЕ ОТОБРАЖАЛСЯ В СПИСКЕ МАКРОСОВ?

Объявите процедуру с помощью ключевого слова `Private`.

```
Private Sub MyMacro()
```

Также можно добавить необязательный аргумент определенного типа данных.

```
Sub MyMacro (Optional FakeArg as Integer)
```

СУЩЕСТВУЕТ ЛИ ВОЗМОЖНОСТЬ СОХРАНЕНИЯ ДИАГРАММЫ В ФАЙЛЕ GIF?

Да. Приведенный ниже код сохраняет первую встроенную диаграмму рабочего листа `Лист1` в виде файла формата GIF с называнием `Mychart.gif`.

```
Set CurrentChart = Sheets("Лист1").ChartObjects(1).Chart
Fname = ThisWorkbook.Path & "\Mychart.gif"
CurrentChart.Export Filename:=Fname, FilterName:="GIF"
```

ДОСТУПНЫ ЛИ ПЕРЕМЕННЫЕ ОДНОЙ ПРОЦЕДУРЫ VBA ДРУГИМ ПРОЦЕДУРАМ VBA? ЧТО, ЕСЛИ ПРОЦЕДУРА НАХОДИТСЯ В ДРУГОМ МОДУЛЕ ИЛИ ДАЖЕ В ДРУГОЙ РАБОЧЕЙ КНИГЕ?

В данном случае речь идет об *области действия переменной*. Существует три типа действия: локальная, в пределах модуля и общедоступная (глобальная). Локальные переменные имеют самую узкую область действия и объявляются внутри процедуры. Локальная переменная используется только в пределах процедуры, в которой она объявлена. Переменные с областью действия уровня модуля объявляются в начале модуля, до объявления первой процедуры. Переменные с областью действия уровня модуля используются во всех процедурах, которые определены в этом модуле. Общедоступные переменные имеют самую широкую область действия и объявляются с помощью ключевого слова `Public`.

Функции

Я СОЗДАЛ ПОЛЬЗОВАТЕЛЬСКУЮ ФУНКЦИЮ РАБОЧЕГО ЛИСТА. ПРИ ПОЛУЧЕНИИ ДОСТУПА К ЭТОЙ ФУНКЦИИ ИЗ ДИАЛОГОВОГО ОКНА МАСТЕР ФУНКЦИЙ ВЫВОДИТСЯ СООБЩЕНИЕ О НЕВОЗМОЖНОСТИ ПРЕДОСТАВЛЕНИЯ СПРАВОЧНОЙ ИНФОРМАЦИИ. КАК ЗАСТАВИТЬ ДИАЛОГОВОЕ ОКНО МАСТЕР ФУНКЦИЙ ОТОБРАЖАТЬ ОПИСАНИЕ ФУНКЦИИ?

Для того чтобы добавить к функции описание, активизируйте рабочую книгу, которая содержит эту функцию. Выберите Сервис⇨Макрос⇨Макросы для отображения диалогового окна Макрос. Функция не указывается в списке, поэтому ее название

необходимо вручную ввести в поле Имя макроса. После ввода имени функции щелкните на кнопке Параметры для отображения диалогового окна Параметры макроса. Введите текст описания в поле Описание.

Можно ли отображать справочную информацию для аргументов пользовательской функции в диалоговом окне Мастер функций?

К сожалению, нет.

Моя функция рабочего листа отображается в категории Определенные пользователем в диалоговом окне Мастер функций. Как переместить функцию в другую категорию?

Для этого необходимо воспользоваться кодом VBA. Следующий оператор перемещает функцию `MyFunc` в категорию 1 (Финансовые функции).

```
Application.MacroOptions Macro:="MyFunc", Category:=1
```

Представленная ниже таблица содержит список допустимых номеров категорий функций.

Номер	Категория
0	Без категории (отображается только в категории Все функции)
1	Финансовые
2	Дата и время
3	Мат. и тригонометрия
4	Статистические
5	Ссылки и массивы
6	Работа с базой данных
7	Текстовые
8	Логические
9	Информационные
10	Команды (обычно эта категория скрыта)
11	Настройка (обычно эта категория скрыта)
12	Управление макросами (обычно эта категория скрыта)
13	DDE/External (обычно эта категория скрыта)
14	Определенные пользователем (по умолчанию)
15	Инженерные (эта категория может использоваться только тогда, когда установлена надстройка Пакет анализа)

Как создать новую категорию функций?

Для этого воспользуйтесь макросом XLM. Однако данный метод ненадежный, поэтому использовать его не рекомендуется.

У меня есть пользовательская функция, которая будет применяться в формуле рабочего листа. Как сделать так, чтобы при вводе неверного значения функция возвращала значение ошибки (#ЗНАЧ!)?

Если функция называется `MyFunction`, то можно использовать следующий оператор для возврата кода ошибки в ячейку, содержащую функцию.

```
MyFunction = CVErr(xlErrValue)
```

В этом примере xlErrValue является предопределенной константой. Константы остальных кодов ошибок описываются в интерактивном справочном руководстве.

КАК ПРОВЕСТИ ПЕРЕСЧЕТ ФОРМУЛ, ПРИМЕНЯЮЩИХ ПОЛЬЗОВАТЕЛЬСКУЮ ФУНКЦИЮ?

Для этого можно воспользоваться комбинацией клавиш <Ctrl+Alt+F9>.

МОЖНО ЛИ ИСПОЛЬЗОВАТЬ ВСТРОЕННЫЕ ФУНКЦИИ EXCEL РАБОЧЕГО ЛИСТА В КОДЕ VBA?

В большинстве случаев — да. Функции Excel рабочего листа задаются с помощью метода WorksheetFunction объекта Application. Например, доступ к функции POWER (СТЕПЕНЬ) вы получите, введя следующий оператор.

```
Ans = Application.WorksheetFunction.Power(5, 3)
```

В этом примере 5 возводится в третью степень, и возвращается результат (125).

Если VBA содержит эквивалент функции рабочего листа, то последнюю нельзя использовать на рабочем листе. Например, если VBA задает функцию подсчета квадратного корня (Sqr), то функцию SQRT рабочего листа в коде VBA использовать нельзя.

EXCEL 95 НЕ ПОДДЕРЖИВАЕТ МЕТОД WORKSHEETFUNCTION. ЗНАЧИТ ЛИ ЭТО, ЧТО МОЕ ПРИЛОЖЕНИЕ ДЛЯ EXCEL 2003 НЕ СОВМЕСТИМО С EXCEL 95?

Нет. На самом деле метод WorksheetFunction использовать не обязательно. Следующие два оператора выполняют одно и то же действие.

```
Ans = Application.WorksheetFunction.Power(5, 3)
Ans = Application.Power(5, 3)
```

МОЖНО ЛИ В КОДЕ VBA ИСПОЛЬЗОВАТЬ ФУНКЦИИ НАДСТРОЙКИ ПАКЕТ АНАЛИЗА?

Да, но для этого необходимо выполнить дополнительные действия. В Excel выберите Сервис⇨Надстройки и установите флажок для надстройки Analysis ToolPak — VBA. После этого активизируйте проект VBA и выберите Tools⇨References (Сервис⇨ Ссылки). Установите флажок для файла atpvbaen.xls, чтобы создать необходимую ссылку. После этого в коде можно использовать любую из функций пакета анализа. Например, приведенный ниже оператор использует функцию CONVERT надстройки Пакет анализа, чтобы преобразовать 5000 метров в мили.

```
MsgBox CONVERT(5000, "m", "mi")
```

СУЩЕСТВУЕТ ЛИ СПОСОБ ЗАДАТЬ РАЗРЫВ СТРОКИ В ТЕКСТЕ ОКНА СООБЩЕНИЯ?

Для этого необходимо воспользоваться символом возврата каретки или перевода строки. Представленный далее оператор отображает окно сообщения, которое содержит текст, состоящий из двух строк. vbCr является встроенной константой, которая представляет символ возврата каретки.

```
MsgBox "Hello" & vbCr & Application.UserName
```

Объекты, свойства, методы и события

Я НЕ ПОНИМАЮ КОНЦЕПЦИЮ ОБЪЕКТОВ. СУЩЕСТВУЕТ ЛИ СПИСОК ОБЪЕКТОВ EXCEL, КОТОРЫЕ МОЖНО ИСПОЛЬЗОВАТЬ?

Да. Диалоговое справочное руководство представляет всю необходимую информацию в удобном графическом формате.

Я ПОТРЯСЕН КОЛИЧЕСТВОМ ДОСТУПНЫХ СВОЙСТВ И МЕТОДОВ. КАК МОЖНО ОПРЕДЕЛИТЬ, КАКИЕ МЕТОДЫ ИЛИ СВОЙСТВА ДОСТУПНЫ ДЛЯ ОПРЕДЕЛЕННОГО ОБЪЕКТА?

Существует несколько способов получения этой информации. Можно воспользоваться окном Object Browser, которое отображается в редакторе VBE. Нажмите клавишу <F2> для получения доступа к окну Object Browser и выберите Excel из раскрывающегося списка Libraries/Workbooks (Все библиотеки). Список Classes (Классы) (слева) содержит все доступные объекты Excel. При выборе объекта его свойства и методы отображаются справа в списке Members (Компоненты).

Кроме того, список свойств и методов можно отобразить при введении имени объекта. Например, введите следующий код.

```
Range("A1").
```

При вводе точки будет отображен список всех доступных свойств и методов объекта `Range`. Если этого не произошло, то необходимо выбрать Tools⇨Options, перейти на вкладку Editor и установить флажок опции Auto List Members.

Также достаточно большое количество информации по языку VBA содержит диалоговое справочное руководство. В нем вы найдете описание методов и свойств всех объектов, которые могут вызвать интерес разработчика. Самым простым способом получения этой информации является ввод имени объекта в поле Immediate в нижней части окна VBE или наведение указателя на имя объекта с последующим нажатием клавиши <F1>. В результате будет отображен раздел справочного руководства, посвященного этому объекту.

В ЧЕМ СМЫСЛ ИСПОЛЬЗОВАНИЯ КОЛЛЕКЦИЙ? ЯВЛЯЕТСЯ ЛИ КОЛЛЕКЦИЯ ОБЪЕКТОМ? ЧТО ТАКОЕ КОЛЛЕКЦИЯ?

Коллекция — это объект, который содержит группу подобных объектов. Коллекция получает название в виде существительного во множественном числе. Например, коллекция `Worksheets` — это объект, который содержит объекты `Worksheet` рабочих книг. Коллекцию можно воспринимать как массив: `Worksheets(1)` указывает на первый объект `Worksheet` в коллекции `Workbooks`. Вместо индекса, допускается указывать имя рабочего листа, например, `Worksheets("Лист1")`. Концепция коллекции упрощает одновременное управление подобными объектами. А для циклического перебора членов коллекции можно воспользоваться конструкцией `For Each-Next`.

КОГДА В КОДЕ VBA Я ССЫЛАЮСЬ НА РАБОЧИЙ ЛИСТ, ВОЗНИКАЕТ СООБЩЕНИЕ ОБ ОШИБКЕ "SUBSCRIPT OUT OF RANGE". НО Я НЕ ИСПОЛЬЗУЮ ПОДСЦЕНАРИИ. ЧТО ПРОИСХОДИТ?

Эта ошибка возникает при попытке получить доступ к несуществующему члену коллекции. Например, приведенный ниже оператор создает сообщение об ошибке, если активная рабочая книга не содержит рабочий лист, который называется `MySheet`.

```
Set X = ActiveWorkbook.Worksheets("MySheet")
```

КАК ЛИШИТЬ ПОЛЬЗОВАТЕЛЯ ВОЗМОЖНОСТИ ПРОКРУЧИВАТЬ РАБОЧИЙ ЛИСТ?

Для этого можно или скрыть неиспользуемые строки и столбцы, или воспользоваться оператором VBA для перенастройки полосы прокрутки на рабочем листе. Следующий оператор устанавливает область прокрутки на листе `Лист1` таким образом, что пользователь не может активизировать ячейки за пределами диапазона `B2:D50`.

```
Worksheets("Лист1").ScrollArea = "B2:D50"
```

Для того чтобы восстановить первоначальную область прокрутки, воспользуйтесь таким оператором.

```
Worksheets("Лист1").ScrollArea = ""
```

Помните, что значение свойства `ScrollArea` не сохраняется вместе с рабочей книгой. Таким образом, необходимо устанавливать это свойство при каждом следующем открытии рабочей книги. Данный оператор можно разместить в процедуре `Workbook_Open`.

КАКАЯ РАЗНИЦА МЕЖДУ ИСПОЛЬЗОВАНИЕМ МЕТОДОВ SELECT И APPLICATION.GOTO?

Метод `Select` объекта `Range` выделяет диапазон только на активном рабочем листе. Метод `Application.Goto` можно использовать для выделения диапазона на любом листе рабочей книги. Метод `Application.Goto` может либо активизировать, либо деактивизировать другой лист. Кроме того, метод `Goto` позволяет прокручивать рабочий лист, чтобы расположить выделенный диапазон в левом верхнем углу экрана.

КАКАЯ РАЗНИЦА МЕЖДУ АКТИВИЗАЦИЕЙ И ВЫДЕЛЕНИЕМ ДИАПАЗОНА?

В отдельных случаях методы `Activate` и `Select` приводят к одинаковому эффекту. Но зачастую они возвращают разные результаты. Предположим, что выделен диапазон `A1:C3`. Приведенный ниже оператор активизирует ячейку `C3`. Первоначальный диапазон остается выделенным, но ячейка `C3` становится активной, т.е. курсор находится в этой ячейке.

```
Range("C3").Activate
```

Предположим также, что диапазон `A1:C3` выделен. Представленный далее оператор выделяет единственную ячейку, которая становится активной.

```
Range("C3").Select
```

СУЩЕСТВУЕТ ЛИ БЫСТРЫЙ СПОСОБ УДАЛЕНИЯ ВСЕХ ЗНАЧЕНИЙ ИЗ РАБОЧЕГО ЛИСТА БЕЗ УДАЛЕНИЯ ФОРМУЛ?

Да. Код выполняется по отношению к активному рабочему листу и удаляет значения всех ячеек, которые не содержат формул (форматирование ячеек тоже не очищается).

```
On Error Resume Next
Cells.SpecialCells(xlCellTypeConstants, 23).ClearContents
```

Использование оператора `On Error Resume Next` позволяет отменить сообщение об ошибке, которое возникает, если ни одна ячейка не соответствует указанному критерию.

Я ЗНАЮ, КАК ПИСАТЬ ОПЕРАТОРЫ VBA ДЛЯ ВЫДЕЛЕНИЯ ДИАПАЗОНА, ДЛЯ КОТОРОГО ИЗВЕСТЕН АДРЕС. ОБЪЯСНИТЕ, КАК СОЗДАТЬ ОПЕРАТОР ВЫДЕЛЕНИЯ, ЕСЛИ ИЗВЕСТНЫ ТОЛЬКО НОМЕР СТРОКИ И НОМЕР СТОЛБЦА?

Для этого воспользуйтесь методом `Cells`. Следующий оператор выделяет ячейку в пятой строке и двенадцатом столбце (т.е. ячейку `L5`).

```
Cells(5, 12).Select
```

СУЩЕСТВУЕТ ЛИ КОМАНДА VBA ДЛЯ ВЫХОДА ИЗ EXCEL? КОГДА Я ПЫТАЮСЬ ЗАПИСАТЬ МАКРОС С КОМАНДОЙ ФАЙЛ⇨ВЫХОД, EXCEL ЗАКРЫВАЕТСЯ ДО ТОГО, КАК ПРЕДОСТАВЛЯЕТСЯ ВОЗМОЖНОСТЬ ПРОСМОТРЕТЬ ЗАПИСАННЫЙ КОД!

Для выхода из Excel необходимо воспользоваться следующим оператором.

```
Application.Quit
```

КАК ОТКЛЮЧИТЬ ОБНОВЛЕНИЕ ЭКРАНА В ПРОЦЕССЕ РАБОТЫ МАКРОСА?

Представленный далее оператор отключает обновление экрана и ускоряет работу макроса, модифицирующего отображаемую информацию.

```
Application.ScreenUpdating = False
```

КАКОЙ СПОСОБ СОЗДАНИЯ ИМЕНИ ДИАПАЗОНА С ПОМОЩЬЮ VBA ЯВЛЯЕТСЯ САМЫМ ПРОСТЫМ?

Если включить запись макроса при создании имени диапазона, то будет получен следующий код.

```
Range("D14:G20").Select
ActiveWorkbook.Names.Add Name:="InputArea", _
    RefersToR1C1:="=Лист1!R14C4:R20C7"
```

Однако намного проще использовать такой оператор.

```
Sheets("Лист1").Range("D14:G20").Name = "InputArea"
```

КАК ОПРЕДЕЛИТЬ, ИМЕЕТ ЛИ ИМЯ КОНКРЕТНАЯ ЯЧЕЙКА ИЛИ ДИАПАЗОН?

Для этого необходимо проверить значение свойства `Name` объекта `Name`, который содержится в объекте `Range`. Следующая функция принимает диапазон в качестве аргумента и возвращает имя диапазона, если оно существует. Если диапазон не имеет имени, то функция возвращает значение `False`.

```
Function RangeName(rng) As Variant
    On Error Resume Next
    RangeName = rng.Name.Name
    If Err <> 0 Then RangeName = False
End Function
```

СУЩЕСТВУЕТ ЛИ СПОСОБ ОТКЛЮЧИТЬ КНОПКИ СТРАНИЦА И ПОЛЯ, КОТОРЫЕ ОТОБРАЖАЮТСЯ В ОКНЕ ПРЕДВАРИТЕЛЬНЫЙ ПРОСМОТР?

Да. Для этого необходимо воспользоваться следующим оператором.

```
ActiveSheet.PrintPreview EnableChanges:=False
```

Аргумент `EnableChanges` метода `PrintPreview` не документирован в справочном руководстве, но он отображается в окне Object Browser.

МОЖНО ЛИ ОТОБРАЖАТЬ СООБЩЕНИЯ В СТРОКЕ СОСТОЯНИЯ В ПРОЦЕССЕ ВЫПОЛНЕНИЯ МАКРОСА? Я ИСПОЛЬЗУЮ СЛОЖНЫЙ МАКРОС И НЕПЛОХО БЫ ИМЕТЬ ПРЕДСТАВЛЕНИЕ О ХОДЕ ЕГО ВЫПОЛНЕНИЯ

Рекомендуется назначить строковые данные свойству `StatusBar` объекта `Application`. Приведем пример соответствующего оператора.

```
Application.StatusBar = "Обработка файла " & FileNum
```

После завершения процедуры необходимо восстановить строку состояния с помощью такого оператора.

```
Application.StatusBar = False
```

Я ЗАПИСАЛ МАКРОС VBA, КОТОРЫЙ КОПИРУЕТ ДИАПАЗОН И ВСТАВЛЯЕТ ЕГО В ДРУГОЕ МЕСТО НА РАБОЧЕМ ЛИСТЕ. ЭТОТ МАКРОС ИСПОЛЬЗУЕТ МЕТОД SELECT. СУЩЕСТВУЕТ ЛИ БОЛЕЕ ЭФФЕКТИВНЫЙ СПОСОБ КОПИРОВАНИЯ И ВСТАВКИ?

Да. Хотя при записи макроса ячейки сначала выделяются, выделение выполнять вовсе не обязательно — на самом деле это только замедляет работу макроса. Запись простой операции копирования и вставки приводит к генерированию кода VBA, состоящего из четырех строк. Две из них вызывают метод `Select`.

```
Range("A1").Select
Selection.Copy
Range("B1").Select
ActiveSheet.Paste
```

Эти четыре строки кода можно заменить одним оператором, который приводится ниже.

```
Range("A1").Copy Range("B1")
```

Обратите внимание, что последний оператор не требует использования метода `Select`.

Я НЕ НАШЕЛ МЕТОДА СОРТИРОВКИ МАССИВА VBA. ЗНАЧИТ ЛИ ЭТО, ЧТО НЕОБХОДИМО КОПИРОВАТЬ ЗНАЧЕНИЯ НА РАБОЧИЙ ЛИСТ И ИСПОЛЬЗОВАТЬ МЕТОД RANGE.SORT?

В языке VBA не существует встроенных средств сортировки массивов. Копирование массива на рабочий лист — это только один из методов, но можно создать собственную процедуру сортировки. В настоящее время существует довольно много алгоритмов сортировки, часть из которых достаточно просто реализуется с помощью VBA. В этой книге приводится код VBA нескольких методов сортировки.

МОЙ МАКРОС РАБОТАЕТ С ВЫДЕЛЕННЫМИ ЯЧЕЙКАМИ, НО ОН НЕУДАЧНО ЗАВЕРШАЕТСЯ, ЕСЛИ ВЫДЕЛЕННОЙ ОКАЗЫВАЕТСЯ ДИАГРАММА. КАК УДОСТОВЕРИТЬСЯ, ЧТО ВЫДЕЛЕН ДИАПАЗОН ЯЧЕЕК?

Можно воспользоваться функцией VBA `TypeName`, которая применяется для проверки объекта `Selection`. Ниже приведен пример такого кода.

```
If TypeName(Selection) <> "Range" Then
    MsgBox "Выделите диапазон!"
    Exit Sub
End If
```

Еще одним вариантом является использование свойства `RangeSelection`, которое возвращает объект `Range`, представляющий выделенные ячейки рабочего листа. Данное свойство возвращает выделенные ячейки на рабочем листе указанного окна, даже если помимо них выделен графический объект. Это свойство объекта `Window`, а не объекта `Workbook`. Следующий оператор отображает адрес выделенного диапазона.

```
MsgBox ActiveWindow.RangeSelection.Address
```

КАК ОПРЕДЕЛИТЬ, АКТИВНА ЛИ ДИАГРАММА

Используйте следующее выражение.

```
If ActiveChart Is Nothing Then MsgBox "Выберите диаграмму!"
```

Окно сообщения появится на экране только в том случае, если диаграмма неактивна (имеется ввиду как внедренная диаграмма, так и диаграмма на отдельных листах).

МАКРОС VBA ДОЛЖЕН ПОДСЧИТАТЬ КОЛИЧЕСТВО СТРОК, ВЫДЕЛЕННЫХ ПОЛЬЗОВАТЕЛЕМ. МЕТОД SELECTION.ROWS.COUNT НЕ ПРИМЕНЯЕТСЯ, ЕСЛИ ВЫДЕЛЕНЫ НЕСМЕЖНЫЕ СТРОКИ. ЭТО ОШИБКА?

На самом деле так и должно быть. Метод `Count` возвращает количество элементов в *первой* выделенной области (предположим, что выделение состоит из нескольких областей). Для того чтобы получить точное количество строк, код VBA должен определить количество выделенных областей и подсчитать количество строк в каждой области. Сначала необходимо воспользоваться методом `Selection.Area.Count`, чтобы получить количество областей. Ниже приведен пример, который сохраняет общее количество выделенных строк в переменной `NumRows`.

```
NumRows = 0
For Each areaCounter In Selection.Areas
    NumRows = NumRows + areaCounter.Rows.Count
Next areaCounter
```

Кстати, это относится и к подсчету количества выделенных столбцов.

Я ИСПОЛЬЗУЮ EXCEL ДЛЯ СОЗДАНИЯ НАКЛАДНЫХ. СУЩЕСТВУЕТ ЛИ СПОСОБ ГЕНЕРАЦИИ УНИКАЛЬНОГО НОМЕРА НАКЛАДНОЙ?

Одним из способов является использование системного реестра. Следующий код демонстрирует такой способ.

```
Counter = GetSetting("XYZ Corp", "InvoiceNum", "Count", 0)
Counter = Counter + 1
SaveSetting "XYZ Corp", "InvoiceNum", "Count", Counter
```

При выполнении данного кода текущее значение извлекается из системного реестра, увеличивается на единицу и назначается переменной Counter. После этого обновленное значение сохраняется в системном реестре. Значение переменной Counter можно использовать в качестве уникального номера накладной.

СУЩЕСТВУЕТ ЛИ СВОЙСТВО РАБОЧЕЙ КНИГИ EXCEL, КОТОРОЕ ПОЗВОЛЯЕТ ЕЙ ВСЕГДА ОТОБРАЖАТЬСЯ НА ЭКРАНЕ И НЕ ПЕРЕКРЫВАТЬСЯ ОКНАМИ ДРУГИХ ПРИЛОЖЕНИЙ?

К сожалению, не существует.

СУЩЕСТВУЕТ ЛИ СПОСОБ БЛОКИРОВКИ СООБЩЕНИЯ EXCEL, ОТОБРАЖАЕМОГО В ПРОЦЕССЕ РАБОТЫ МАКРОСА? НАПРИМЕР, МНЕ НЕОБХОДИМО ИЗБАВИТЬСЯ ОТ СООБЩЕНИЯ, ВЫВОДИМОГО НА ЭКРАН ПРИ УДАЛЕНИИ ДАННЫХ МАКРОСОМ РАБОЧЕГО ЛИСТА.

Следующий оператор отключает большинство предупреждений Excel:

```
Application.DisplayAlerts = False
```

СУЩЕСТВУЕТ ЛИ ОПЕРАТОР VBA ДЛЯ ВЫДЕЛЕНИЯ ПОСЛЕДНЕГО ЗНАЧЕНИЯ В СТРОКЕ ИЛИ В СТОЛБЦЕ? КАК ПРАВИЛО, Я МОГУ ВОСПОЛЬЗОВАТЬСЯ КОМБИНАЦИЕЙ КЛАВИШ <CTRL+SHIFT+СТРЕЛКА ВНИЗ> ИЛИ <CTRL+SHIFT+СТРЕЛКА ВВЕРХ> ДЛЯ ВЫПОЛНЕНИЯ ЭТОГО ДЕЙСТВИЯ, НО КАК ЭТО СДЕЛАТЬ С ПОМОЩЬЮ МАКРОСА?

Эквивалентом комбинации клавиш <Ctrl+Shift+стрелка вниз> в VBA выступает оператор

```
Selection.End(xlDown).Select
```

Для других направлений используется три других константы: xlToLeft, xlToRight и xlUp.

КАК ОПРЕДЕЛИТЬ ПОСЛЕДНЮЮ НЕПУСТУЮ ЯЧЕЙКУ В ВЫБРАННОМ СТОЛБЦЕ?

Представленный ниже оператор отображает адрес последней непустой ячейки в столбце A.

```
MsgBox ActiveSheet.Range("A65536").End(xlUp).Address
```

Следующий оператор выполняется, если рабочий лист имеет больше 65536 строк.

```
MsgBox ActiveSheet.Cells(Rows.Count, 1).End(xlUp).Address
```

НО ЭТОТ ОПЕРАТОР НЕ БУДЕТ РАБОТАТЬ, ЕСЛИ ЯЧЕЙКА A65536 СОДЕРЖИТ ЗНАЧЕНИЕ!

Для того чтобы обойти такую маловероятную ситуацию, необходимо воспользоваться следующим кодом.

```
With ActiveSheet.Cells(Rows.Count, 1)
    If IsEmpty(.Value) Then
        MsgBox .End(xlUp).Address
    Else
        MsgBox .Address
    End If
End With
```

ССЫЛКИ VBA МОГУТ СТАТЬ ОЧЕНЬ ДЛИННЫМИ, ОСОБЕННО ЕСЛИ НЕОБХОДИМО ТОЧНО УКАЗАТЬ ОБЪЕКТ В РАБОЧЕЙ КНИГЕ И РАБОЧЕМ ЛИСТЕ. СУЩЕСТВУЕТ ЛИ СПОСОБ СОКРАТИТЬ ДЛИНУ ССЫЛОК?

Да. Для этого воспользуйтесь оператором `Set` для создания объектной переменной. Ниже приведен пример подобного кода.

```
Dim MyRange as Range
Set MyRange = ThisWorkbook.Worksheets("Лист1").Range("A1")
```

После выполнения оператора `Set` можно ссылаться на объект `Range` по имени `MyRange`. Например, вы имеете возможность определить значение ячейки с помощью следующего оператора.

```
MyRange.Value = 10
```

Кроме упрощения ссылок на объекты, использование таких переменных ускоряет выполнение кода.

СУЩЕСТВУЕТ ЛИ СПОСОБ ОБЪЯВЛЕНИЯ МАССИВА, ЕСЛИ НЕ ИЗВЕСТНО, СКОЛЬКО ЭЛЕМЕНТОВ ОН БУДЕТ СОДЕРЖАТЬ?

Да. Можно объявить динамический массив с помощью оператора `Dim`. При этом используются пустые скобки. Как только станет известно, сколько элементов будет содержаться в массиве, воспользуйтесь оператором `ReDim`, чтобы переопределить массив. Оператор `ReDim Preserve` позволяет переопределить массив без потери текущего содержимого.

МОЖНО ЛИ ПРЕДОСТАВИТЬ ПОЛЬЗОВАТЕЛЮ ВОЗМОЖНОСТЬ ОТМЕНИТЬ РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ МАКРОСА?

Да, но эта возможность не доступна по умолчанию. Для того чтобы включить ее, модуль кода VBA должен отслеживать изменения, проведенные макросом. Макрос должен быть готов отменить внесенные изменения при выборе пользователем команды Правка⇨Отмена.

Чтобы разрешить пользователю выполнять команду Правка⇨Отмена, воспользуйтесь методом `OnUndo` в качестве последнего оператора макроса. Этот метод позволяет указать текст, который отображается в опции меню Отмена, а также процедуру, вызываемую командой Правка⇨Отмена.

```
Application.OnUndo "Последний макрос", "MyUndoMacro"
```

У МЕНЯ ЕСТЬ МАКРОС LOTUS 1-2-3, КОТОРЫЙ ОСТАНАВЛИВАЕТ СВОЮ РАБОТУ ДЛЯ ПРЕДОСТАВЛЕНИЯ ПОЛЬЗОВАТЕЛЮ ВОЗМОЖНОСТИ ВВЕДЕНИЯ ДАННЫХ В ЯЧЕЙКУ. КАК ДОСТИЧЬ ТАКОГО ЖЕ ЭФФЕКТА В МАКРОСЕ VBA?

Excel не может повторить этот тип поведения, однако вы вправе использовать функцию Excel `InputBox`, чтобы получить значение от пользователя и разместить его в определенной ячейке. Первый оператор, который показан ниже, отображает окно ввода. Когда пользователь вводит значение, это значение размещается в ячейке `A1`.

```
UserVal = Application.InputBox(prompt:="Введите значение", Type:=1)
If UserVal <> False Then Range("A1") = UserVal
```

VBA ИМЕЕТ ФУНКЦИЮ INPUTBOX, НО МЕТОД С ТАКИМ ЖЕ НАЗВАНИЕМ СОДЕРЖИТСЯ В ОБЪЕКТЕ APPLICATION. ОНИ ОДИНАКОВЫ?

Нет. Метод Excel `InputBox` является более гибким, так как позволяет проверить введенное пользователем значение. В предыдущем примере в качестве значения аргумента `Type` метода `InputBox` использовалась единица (представляет числовое значение). Это значение позволяет удостовериться, что пользователь ввел в поле числовое значение.

ПРИ ИСПОЛЬЗОВАНИИ ФУНКЦИИ RGB ДЛЯ УКАЗАНИЯ ЦВЕТА ДОВОЛЬНО ЧАСТО ЦВЕТ ОКАЗЫВАЕТСЯ НЕПРАВИЛЬНЫМ. ЧТО Я ДЕЛАЮ НЕВЕРНО?

Возможно, ничего. Рабочая книга Excel может использовать только 56 цветов (палитру из 56 цветов). Если указанный цвет RGB не входит в число цветов палитры, Excel использует самый близкий цвет на поддерживаемой палитре.

Я ПЫТАЮСЬ НАПИСАТЬ ОПЕРАТОР VBA, КОТОРЫЙ СОЗДАЕТ ФОРМУЛУ. ДЛЯ ЭТОГО, КАК ИЗВЕСТНО, НЕОБХОДИМО ВСТАВИТЬ СИМВОЛ КАВЫЧЕК (") В ТЕКСТ. КАК ЭТО СДЕЛАТЬ?

Предположим, что вам с помощью кода VBA необходимо ввести следующую формулу в ячейку с адресом `B1`.

```
=ЕСЛИ(A1="Да";ИСТИНА;ЛОЖЬ)
```

Следующий оператор генерирует сообщение о синтаксической ошибке.

```
Range("B1").Formula = "=IF(A1="Yes",TRUE,FALSE)"
```

Решением будет последовательное использование двух двойных кавычек. Представленный далее оператор приводит к необходимому результату.

```
Range("B1").Formula = "=IF(A1=""Yes"",TRUE,FALSE)"
```

Еще одним способом является использование функции VBA `Chr` с аргументом 34. Функция возвращает символ кавычки. Следующий пример демонстрирует применение этого способа.

```
Range("B1").Formula = _
  "=IF(A1=" & Chr(34) & "Yes" & Chr(34) & ",TRUE,FALSE)"
```

Я СОЗДАЛ МАССИВ, НО ПЕРВЫЙ ЕГО ЭЛЕМЕНТ ВОСПРИНИМАЕТСЯ КАК ВТОРОЙ. ЧТО Я ДЕЛАЮТ НЕПРАВИЛЬНО?

Если не указать обратного, VBA использует значение 0 в качестве индекса первого элемента массива. Если необходимо, чтобы индекс массива начинался с 1, вставьте представленный ниже оператор в начало модуля VBA.

```
Option Base 1
```

Кроме того, можно указать верхнюю и нижнюю границы массива при его создании.

```
Dim Months(1 To 12) As String
```

Я ХОТЕЛ БЫ ДОБИТЬСЯ ОТ КОДА VBA МАКСИМАЛЬНОГО БЫСТРОДЕЙСТВИЯ. ЧТО ВЫ МОЖЕТЕ ПОСОВЕТОВАТЬ?

Предложим несколько общих советов. Удостоверьтесь, что объявлены все используемые переменные. Воспользуйтесь оператором `Option Explicit` в начале модуля, чтобы сделать объявление переменных обязательным. Если ссылка на объект Excel применяется более одного раза, создайте для этого объекта переменную. Используйте конструкцию `With-End With` везде, где это возможно. Если макрос записывает информацию на рабочий лист, отключите обновление экрана с помощью оператора `Application.ScreenUpdating = False`. Если приложение вводит данные в ячейки, на которые ссылается более одной формулы, то следует включить ручной режим пересчета, чтобы избежать лишних вычислений.

Пользовательские диалоговые окна

МНЕ НЕОБХОДИМО ПОЛУЧИТЬ ОТ ПОЛЬЗОВАТЕЛЯ ДАННЫЕ, НО ДИАЛОГОВОЕ ОКНО USERFORM ТРЕБУЕТ СЛОЖНОГО ПРОГРАММИРОВАНИЯ. СУЩЕСТВУЕТ ЛИ АЛЬТЕРНАТИВА?

Да. Обратитесь к функциям VBA `MsgBox` и `InputBox`. Кроме того, можно воспользоваться методом Excel `InputBox`.

ДОПУСТИМ, В ДИАЛОГОВОМ ОКНЕ USERFORM НАХОДИТСЯ 12 ЭЛЕМЕНТОВ УПРАВЛЕНИЯ COMMANDBUTTON. КАК СОЗДАТЬ ЕДИНЫЙ МАКРОС, КОТОРЫЙ БУДЕТ ВЫЗЫВАТЬСЯ ПРИ ЩЕЛЧКЕ НА ЛЮБОЙ ИЗ КНОПОК?

Для решения этой задачи не существует простого способа, так как каждый элемент управления `CommandButton` имеет собственную процедуру `Click`. Одним из решений является вызов дополнительной процедуры из каждой процедуры обработки события `CommandButton_Click`. Другое решение — это создание модуля класса для получения нового объекта (см. главу 15).

СУЩЕСТВУЕТ ЛИ СПОСОБ ОТОБРАЗИТЬ ДИАГРАММУ В ДИАЛОГОВОМ ОКНЕ USERFORM?

Непосредственного способа получения такой диаграммы нет. Одним из решений является использование макроса, который сохраняет диаграмму в виде файла формата GIF и загружает файл GIF в элемент управления `Image`.

КАК УДАЛИТЬ КНОПКУ X ИЗ СТРОКИ ЗАГОЛОВКА ДИАЛОГОВОГО ОКНА USERFORM? МНЕ НЕОБХОДИМО ЛИШИТЬ ПОЛЬЗОВАТЕЛЯ ВОЗМОЖНОСТИ ЗАКРЫВАТЬ ДИАЛОГОВОЕ ОКНО С ПОМОЩЬЮ ЭТОЙ КНОПКИ

Удаление кнопки Закрыть из строки заголовка диалогового окна `UserForm` требует использования нескольких сложных функций Windows API. Более простым решением будет перехват всех попыток закрыть диалоговое окно. Воспользуйтесь процедурой `UserForm_QueryClose`, которая находится в модуле кода диалогового окна `UserForm`. Следующий пример иллюстрирует результат того, что пользователь не может закрыть диалоговое окно с помощью кнопки Закрыть.

```
Private Sub UserForm_QueryClose _
  (Cancel As Integer, CloseMode As Integer)
    If CloseMode = vbFormControlMenu Then
```

```
        MsgBox "Вы не можете закрыть окно этим способом"
        Cancel = True
    End If
End Sub
```

Я СОЗДАЛ ДИАЛОГОВОЕ ОКНО USERFORM, ЭЛЕМЕНТЫ УПРАВЛЕНИЯ КОТОРОГО СВЯЗАНЫ С ЯЧЕЙКАМИ РАБОЧЕГО ЛИСТА С ПОМОЩЬЮ СВОЙСТВА CONTROLSOURCE. СУЩЕСТВУЕТ ЛИ БОЛЕЕ УДАЧНЫЙ СПОСОБ ПОЛУЧЕНИЯ ПОДОБНОГО РЕЗУЛЬТАТА?

Рекомендуется избегать использования ссылок на ячейки рабочего листа, кроме случаев, когда это действительно необходимо. Создание таких ссылок приводит к замедлению работы приложения, так как рабочий лист пересчитывается каждый раз, когда элемент управления вносит изменения в связанную ячейку.

СУЩЕСТВУЕТ ЛИ СПОСОБ СОЗДАНИЯ МАССИВА ЭЛЕМЕНТОВ УПРАВЛЕНИЯ ДЛЯ ДИАЛОГОВОГО ОКНА USERFORM? ЭТО МОЖНО СДЕЛАТЬ В VISUAL BASIC 6.0, НО Я НЕ МОГУ ПОНЯТЬ, КАК ЭТО ДЕЛАЕТСЯ В VBA.

Массив элементов управления создать нельзя, но можно создать массив объектов `Control`. Следующий код создает массив, который содержит элементы управления `CommandButton`.

```
Private Sub UserForm_Initialize()
    Dim Buttons() As CommandButton
    Cnt = 0
    For Each Ctl In UserForm1.Controls
        If TypeName(Ctl) = "CommandButton" Then
            Cnt = Cnt + 1
            ReDim Preserve Buttons(1 To Cnt)
            Set Buttons(Cnt) = Ctl
        End If
    Next Ctl
End Sub
```

СУЩЕСТВУЕТ ЛИ РАЗНИЦА МЕЖДУ СКРЫТИЕМ ДИАЛОГОВОГО ОКНА USERFORM И ЕГО ВЫГРУЗКОЙ ИЗ ПАМЯТИ?

Да. Метод `Hide` оставляет диалоговое окно `UserForm` в памяти, но делает его невидимым. Оператор `Unload` выгружает диалоговое окно, начиная процесс "уничтожения" (вызывая событие `Terminate` объекта `UserForm`) и удаляя диалоговое окно `UserForm` из памяти.

КАК ОБЕСПЕЧИТЬ ОТОБРАЖЕНИЕ ДИАЛОГОВОГО ОКНА USERFORM В МОМЕНТ ВЫПОЛНЕНИЯ ПОЛЬЗОВАТЕЛЕМ ДРУГИХ ДЕЙСТВИЙ?

По умолчанию каждое диалоговое окно `UserForm` отображается в модальном режиме. Это означает, что перед началом других действий диалоговое окно необходимо закрыть. Начиная с Excel 2000, можно создавать немодальные диалоговые окна `UserForm`. Для этого методу `Show` передается аргумент `vbModeless`. Ниже приведен пример такого оператора.

```
UserForm1.Show vbModeless
```

EXCEL ВЫДАЕТ СООБЩЕНИЕ ОБ ОШИБКЕ КОМПИЛЯЦИИ, КОГДА Я ВЫЗЫВАЮ МЕТОД USERFORM1.SHOW С ПАРАМЕТРОМ VBMODELESS. КАК СОЗДАТЬ НЕМОДАЛЬНЫЕ ОКНА В EXCEL 2003, ОСТАВЛЯЯ ИХ МОДАЛЬНЫМИ В EXCEL 97?

Необходимо проверить, какая версия Excel используется для запуска приложения, после чего по результатам проверки запустить соответствующую процедуру. Следующий код демонстрирует этот способ.

```
Sub ShowUserForm()
    If Val(Application.Version) >= 9 Then
      ShowModelessForm
    Else
      UserForm1.Show
    End If
End Sub

Sub ShowModelessForm()
    Dim frm As Object
    Set frm = UserForm1
    frm.Show 0  ' vbModeless
End Sub
```

Так как процедура `ShowModelessForm` не выполняется в Excel 97, она не приведет к появлению ошибки компиляции.

МНЕ НЕОБХОДИМО ОТОБРАЗИТЬ ИНДИКАТОР ТЕКУЩЕГО СОСТОЯНИЯ, ПОКА ПРОИСХОДИТ ДОЛГИЙ ПРОЦЕСС УСТАНОВКИ ПРИЛОЖЕНИЯ. КАК ЭТО СДЕЛАТЬ?

Воспользуйтесь диалоговым окном `UserForm`. В главе 15 описано несколько способов создания индикаторов текущего состояния, в том числе постепенное увеличение длины прямоугольника, в течение выполнения процесса.

КАК ВОСПОЛЬЗОВАТЬСЯ ИНСТРУМЕНТАМИ РИСОВАНИЯ EXCEL, ЧТОБЫ ДОБАВИТЬ ПРОСТЫЕ РИСУНКИ В ДИАЛОГОВОЕ ОКНО USERFORM?

Инструменты рисования нельзя применять непосредственно с пользовательскими диалоговыми окнами, но их можно применять косвенно. Начните с создания рисунка на рабочем листе. После этого выделите рисунок и выберите Правка⇨Копировать. Активизируйте диалоговое окно `UserForm` и вставьте в него объект `Image`. Нажмите клавишу <F4> для отображения окна Properties. Выберите свойство `Picture` и нажмите комбинацию клавиш <Ctrl+V>, чтобы разместить содержимое буфера обмена в элемент управления `Image`. Кроме того, вам может понадобится установить свойство `AutoSize` в значение `True`.

КАК СОЗДАТЬ СПИСОК ФАЙЛОВ И ПАПОК В ДИАЛОГОВОМ ОКНЕ USERFORM, ЧТОБЫ ПОЛЬЗОВАТЕЛЬ МОГ ВЫБРАТЬ ОДИН ИЗ ФАЙЛОВ?

Создавать такое диалоговое окно `UserForm` нет необходимости. Для этого существует метод VBA `GetOpenFilename`. Он отображает диалоговое окно, в котором пользователь может выбрать диск, папку и файл.

У МЕНЯ ЕСТЬ НЕСКОЛЬКО ФАЙЛОВ LOTUS 1-2-3 ДЛЯ WINDOWS И QUATTRO PRO ДЛЯ WINDOWS, КОТОРЫЕ СОДЕРЖАТ ПОЛЬЗОВАТЕЛЬСКИЕ ДИАЛОГОВЫЕ ОКНА. СУЩЕСТВУЕТ ЛИ УТИЛИТА ДЛЯ ПРЕОБРАЗОВАНИЯ ЭТИХ ДИАЛОГОВЫХ ОКОН В ДИАЛОГОВЫЕ ОКНА EXCEL?

Нет, не существует.

МНЕ НЕОБХОДИМО ОБЪЕДИНИТЬ СТРОКИ И ОТОБРАЗИТЬ ИХ В ЭЛЕМЕНТЕ УПРАВЛЕНИЯ LISTBOX. НО ПРИ ВЫПОЛНЕНИИ ЭТОЙ ОПЕРАЦИИ СТРОКИ ОТОБРАЖАЮТСЯ С НЕПРАВИЛЬНЫМ ВЫРАВНИВАНИЕМ. КАК ДОБИТЬСЯ ОДИНАКОВОГО ВЫРАВНИВАНИЯ?

Одним из решений является использование моноширинного шрифта, например Courier New. Но лучше настроить элемент управления `ListBox` на форматирование в два столбца (дополнительная информация приведена в главе 14).

СУЩЕСТВУЕТ ЛИ ВОЗМОЖНОСТЬ ОТОБРАЖЕНИЯ ВСТРОЕННОГО ДИАЛОГОВОГО ОКНА EXCEL С ПОМОЩЬЮ КОДА VBA?

Многие (но не все) диалоговые окна Excel могут быть отображены с помощью метода `Application.Dialogs`. Например, следующий оператор отображает диалоговое окно, которое позволяет форматировать числа в ячейках.

```
Application.Dialogs(xlDialogFormatNumber).Show
```

Для просмотра констант, соответствующих встроенным диалоговым окнам, можно воспользоваться окном Object Browser. В редакторе VBE нажмите клавишу <F2>, выберите библиотеку Excel и найдите опцию `xlDialog`. Скорее всего, для поиска необходимой константы вам придется воспользоваться методом проб и ошибок.

Я ПОПЫТАЛСЯ ПРИМЕНИТЬ МЕТОДИКУ, ОПИСАННУЮ В ПРЕДЫДУЩЕМ ВОПРОСЕ, НО ПОЛУЧИЛ СООБЩЕНИЕ ОБ ОШИБКЕ. ПОЧЕМУ?

Метод `Dialogs` завершится неудачно, если его вызвать в неподходящем контексте. Например, если попытаться отобразить диалоговое окно Тип диаграммы (`xlDialogChartType`), когда диаграмма не активизирована, то будет выведено сообщение об ошибке.

КАЖДЫЙ РАЗ ПРИ СОЗДАНИИ ДИАЛОГОВОГО ОКНА USERFORM Я ДОБАВЛЯЮ КНОПКИ ОК И ОТМЕНА. СУЩЕСТВУЕТ ЛИ СПОСОБ АВТОМАТИЧЕСКОГО ДОБАВЛЕНИЯ ЭТИХ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ?

Да. Настройте диалоговое окно `UserForm`, чтобы оно содержало все элементы управления, которые используются наиболее часто. После этого выберите File⇨Export File (Файл⇨Экспорт файла) для сохранения этого диалогового окна. Когда возникнет необходимость добавить новое диалоговое окно в проект, выберите команду File⇨Import File (Файл⇨Импорт файла).

СУЩЕСТВУЕТ ЛИ ВОЗМОЖНОСТЬ СОЗДАНИЯ ДИАЛОГОВОГО ОКНА USERFORM БЕЗ СТРОКИ ЗАГОЛОВКА?

Нет. Самое большее, что можно сделать, — это очистить строку заголовка, указав в качестве значения его свойства `Caption` пустую строку.

Я ЗАПИСАЛ МАКРОС VBA, ПРЕДНАЗНАЧЕННЫЙ ДЛЯ ВЫПОЛНЕНИЯ ОПЕРАЦИИ ПЕЧАТИ В ФАЙЛ. НО В КОДЕ ОТСУТСТВУЕТ СПОСОБ УКАЗАНИЯ ИМЕНИ ФАЙЛА И ВЫДАЕТСЯ СООБЩЕНИЕ О НЕОБХОДИМОСТИ ВВЕСТИ ИМЯ ФАЙЛА.

Эта распространенная проблема была решена в Excel 2000. В Excel 2000 и более поздних версиях можно указать аргумент `PrToFileName` метода `PrintOut`. Ниже приведен пример использования данного метода.

```
ActiveSheet.PrintOut PrintToFile:=True, _
  PrToFileName:="test.prn"
```

КОГДА Я ЩЕЛКАЮ НА КНОПКЕ В ДИАЛОГОВОМ ОКНЕ USERFORM, НИЧЕГО НЕ ПРОИСХОДИТ. ЧТО Я ДЕЛАЮ НЕПРАВИЛЬНО?

Элементы управления, добавленные в диалоговое окно `UserForm`, не выполняют никакой функции, пока им не будет предоставлена процедура обработки события. Эти процедуры должны располагаться в модуле кода диалогового окна `UserForm`, а также иметь соответствующие имена.

МОГУ ЛИ Я СОЗДАТЬ ДИАЛОГОВОЕ ОКНО, РАЗМЕР КОТОРОГО БУДЕТ ОСТАВАТЬСЯ ПОСТОЯННЫМ, НЕЗАВИСИМО ОТ ТЕКУЩЕГО РАЗРЕШЕНИЯ ЭКРАНА?

Да, такое диалоговое окно можно создать, но стоит ли этот результат затраченных усилий? Рекомендуется создать код, который определяет разрешение экрана и использует свойство `Zoom` диалогового окна `UserForm` для изменения его размера. Чтобы избежать подобной проблемы, все диалоговые окна разрабатываются для разрешения 640×480.

СУЩЕСТВУЕТ ЛИ ВОЗМОЖНОСТЬ СОЗДАНИЯ ДИАЛОГОВОГО ОКНА USERFORM, КОТОРОЕ ПОЗВОЛЯЕТ УКАЗЫВАТЬ ДИАПАЗОН ЯЧЕЕК С ПОМОЩЬЮ МЫШИ?

Да. Воспользуйтесь элементом управления `RefEdit`. Пример использования этого элемента управления приводится в главе 14.

СУЩЕСТВУЕТ ЛИ СПОСОБ ИЗМЕНЕНИЯ РАСПОЛОЖЕНИЯ ДИАЛОГОВОГО ОКНА USERFORM?

Существует. Необходимо лишь установить значения свойств `Left` и `Top` диалогового окна `UserForm`. Но чтобы изменение было реализовано, установите свойство `StartUpPosition` в значение 0.

МОЖНО ЛИ ДОБАВИТЬ В РАБОЧУЮ КНИГУ ДИАЛОГОВЫЙ ЛИСТ EXCEL 5/95?

Да. Щелкните правой кнопкой мыши на любом ярлыке листа в рабочей книге и выберите из контекстного меню опцию Добавить. В диалоговом окне Вставка укажите Окно диалога Excel 5.0. Но помните, что в этой книге не приводится информация о диалоговых листах Excel 5/95.

Надстройки

ГДЕ МОЖНО НАЙТИ НАДСТРОЙКИ ДЛЯ EXCEL?

Надстройки для Excel расположены в нескольких источниках.

- В Excel интегрировано несколько надстроек, которые можно использовать в любой момент.
- Дополнительные надстройки можно загрузить с узла Microsoft Office Update.
- Независимые производители создают надстройки, предназначенные для выполнения самых разных задач.
- Некоторые разработчики создают бесплатные надстройки и распространяют их через Web-узлы.
- Можно создать собственные надстройки.

КАК УСТАНОВИТЬ НАДСТРОЙКУ?

Для этого загрузите надстройку с помощью команды Сервис⇨Надстройки или команды Файл⇨Открыть. Использовать команду Сервис⇨Надстройки предпочтительнее, поскольку надстройка, открытая с помощью команды Файл⇨Открыть, не может быть закрыта с помощью VBA.

ПРИ УСТАНОВКЕ СОБСТВЕННОЙ НАДСТРОЙКИ С ПОМОЩЬЮ ДИАЛОГОВОГО ОКНА НАДСТРОЙКИ Я НЕ ОБНАРУЖИЛ ЕЕ ИМЕНИ И ОПИСАНИЯ. КАК ДОБАВИТЬ ОПИСАНИЕ К НАДСТРОЙКЕ?

Перед созданием надстройки необходимо воспользоваться командой Файл⇨Свойства, чтобы открыть диалоговое окно Свойства. Перейдите на вкладку Документ. В поле Название введите текст, который должен отображаться в диалоговом окне Надстройки. В поле Заметки введите описание надстройки. После этого можно продолжать создавать надстройку.

У МЕНЯ ЕСТЬ НЕСКОЛЬКО НАДСТРОЕК, КОТОРЫЕ Я БОЛЬШЕ НЕ ИСПОЛЬЗУЮ, НО Я НИКАК НЕ МОГУ РАЗОБРАТЬСЯ, КАК УДАЛИТЬ ИХ ИЗ СПИСКА В ДИАЛОГОВОМ ОКНЕ НАДСТРОЙКИ. В ЧЕМ ПРОБЛЕМА?

Странно, но неиспользуемые надстройки невозможно удалить из списка средствами Excel. Необходимо отредактировать содержимое системного реестра и удалить ссылки на файлы надстроек, которые больше не должны использоваться. Еще один способ удаления надстроек заключается в удалении файлов, в которых они хранятся. После этого при попытке открыть надстройку из диалогового окна Надстройки Excel предложит удалить надстройку из списка.

КАК СОЗДАТЬ НАДСТРОЙКУ?

Активизируйте рабочий лист и выполните команду Файл⇨Сохранить как. После этого из раскрывающегося списка Тип файла выберите Надстройка Microsoft Excel (*.xla).

Я ПЫТАЮСЬ СОЗДАТЬ НАДСТРОЙКУ, НО ДИАЛОГОВОЕ ОКНО СОХРАНЕНИЕ ДОКУМЕНТА НЕ ПРЕДОСТАВЛЯЕТ ВОЗМОЖНОСТИ ВЫБРАТЬ НЕОБХОДИМЫЙ ТИП ФАЙЛА.

Скорее всего, активный лист не является рабочим листом.

НУЖНО ЛИ ПРЕОБРАЗОВЫВАТЬ ВСЕ КЛЮЧЕВЫЕ РАБОЧИЕ КНИГИ В НАДСТРОЙКИ?

Нет! Несмотря на то, что надстройку можно создать на основе любой рабочей книги, не все рабочие книги подходят на эту роль. Когда рабочая книга превращается в надстройку, она становится полностью невидимой. Для большинства рабочих книг невидимость означает невозможность использования.

СУЩЕСТВУЕТ ЛИ НЕОБХОДИМОСТЬ В ХРАНЕНИИ ДВУХ ВЕРСИЙ РАБОЧЕЙ КНИГИ: XLS И XLA?

В версиях до Excel 97 хранение обеих версий было необходимым. Начиная с Excel 97, это делать не обязательно. Надстройку можно легко обратно преобразовывать в рабочую книгу.

КАК ВНЕСТИ ИЗМЕНЕНИЯ В НАДСТРОЙКУ ПОСЛЕ ЕЕ СОЗДАНИЯ?

Запустите редактор VBE (комбинация клавиш <Alt+F11>) и установите свойство `IsAddIn` объекта `ThisWorkbook` в значение `False`. Внесите все необходимые изменения, установите свойство `IsAddIn` в значение `True` и повторно сохраните файл.

КАКАЯ РАЗНИЦА МЕЖДУ ФАЙЛОМ XLS И ФАЙЛОМ XLA, КОТОРЫЙ СОЗДАН НА ОСНОВЕ ФАЙЛА XLS? ОТКОМПИЛИРОВАН ЛИ ФАЙЛ XLA? МОЖЕТ, ОН РАБОТАЕТ БЫСТРЕЕ?

Между данными файлами не существует особой разницы, и увеличение быстродействия несущественно. Код VBA всегда компилируется перед выполнением. Это касается как файлов XLS, так и файлов XLA. Файлы XLA содержат код VBA, а не откомпилированный код.

КАК ЗАЩИТИТЬ КОД НАДСТРОЙКИ ОТ ПРОСМОТРА ПОЛЬЗОВАТЕЛЯМИ?

Запустите редактор VBE и выберите Tools⇨xxx Properties (Сервис⇨Свойства xxx) (где xxx — название проекта). Перейдите на вкладку Protection (Защита) и установите флажок Lock project from viewing (Блокировать просмотр проекта), затем введите пароль.

ЗАЩИЩЕНЫ ЛИ МОИ НАДСТРОЙКИ? ДРУГИМИ СЛОВАМИ, ЕСЛИ Я РАСПРОСТРАНЯЮ ФАЙЛ XLA, МОЖНО ЛИ БЫТЬ УВЕРЕННЫМ, ЧТО БОЛЬШЕ НИКТО НЕ УВИДИТ МОЙ ИСХОДНЫЙ КОД?

Надстройку можно защитить с помощью пароля. Это предотвратит доступ к исходному коду для большинства пользователей. Новые версии Excel имеют улучшенные возможности по обеспечению безопасности, но остается вероятность взлома пароля с помощью целого ряда утилит. Если это вас не устраивает, то обратитесь к другому способу распространения приложений.

Объекты CommandBar

EXCEL 95 СОДЕРЖАЛ УДОБНЫЙ РЕДАКТОР МЕНЮ, НО В EXCEL 97 И БОЛЕЕ ПОЗДНИХ ВЕРСИЯХ ЕГО НЕТ. ПОЧЕМУ?

Начиная с Excel 97, панели инструментов и меню Excel стали полностью другими. Теперь они представляются объектами `CommandBar`. Редактор меню выведен из состава Excel, поэтому для редактирования меню и панелей инструментов необходимо использовать диалоговое окно Настройка (выберите команду Сервис⇨Настройка).

МОЖНО ЛИ РЕДАКТИРОВАТЬ МЕНЮ, КОТОРЫЕ СОЗДАНЫ В РЕДАКТОРЕ МЕНЮ EXCEL 95?

Да, но лишь посредством Excel 95.

ПОСЛЕ РЕДАКТИРОВАНИЯ МЕНЮ С ПОМОЩЬЮ ДИАЛОГОВОГО ОКНА НАСТРОЙКА ИЗМЕНЕНИЯ БЫЛИ СОХРАНЕНЫ И ИСПОЛЬЗУЮТСЯ ПОСТОЯННО. КАК МОЖНО ВНЕСТИ ИЗМЕНЕНИЯ В МЕНЮ ТОЛЬКО ДЛЯ ОДНОЙ РАБОЧЕЙ КНИГИ?

Вносите изменения с помощью кода VBA, когда рабочая книга загружается, и восстанавливайте меню с помощью кода VBA, когда рабочая книга выгружается из памяти.

Я ЗНАЮ, КАК МОЖНО ИСПОЛЬЗОВАТЬ СВОЙСТВО FACEID ДЛЯ ОПРЕДЕЛЕНИЯ ИЗОБРАЖЕНИЯ НА ЭЛЕМЕНТЕ УПРАВЛЕНИЯ. НО КАКОЕ ЗНАЧЕНИЕ СВОЙСТВА FACEID ОТНОСИТСЯ К КОНКРЕТНОМУ ИЗОБРАЖЕНИЮ?

Microsoft не предоставляет способа получения этой информации, но существует несколько утилит, которые позволяют легко идентифицировать значения свойства `FaceId` для изображений. В главе 22 приводится пример полезной надстройки.

Я ПРИСОЕДИНИЛ НОВУЮ ВЕРСИЮ ПАНЕЛИ ИНСТРУМЕНТОВ К РАБОЧЕЙ КНИГЕ, НО EXCEL ПРОДОЛЖАЕТ ОТОБРАЖАТЬ СТАРУЮ ВЕРСИЮ. КАК ЗАСТАВИТЬ EXCEL ИСПОЛЬЗОВАТЬ НОВУЮ ВЕРСИЮ?

Когда Excel открывает рабочую книгу, к которой присоединена панель инструментов, существующая панель инструментов не замещается. Лучшим решением будет использование кода VBA, который создает панель инструментов "на лету" при открытии рабочей книги и удаляет эту панель инструментов при закрытии рабочей книги. Кроме того, можно присоединить панель инструментов к рабочей книге и создать код VBA, который будет удалять панель инструментов при закрытии рабочей книги.

Я ВНЕС БОЛЬШОЕ КОЛИЧЕСТВО ИЗМЕНЕНИЙ В ПАНЕЛИ ИНСТРУМЕНТОВ EXCEL. КАК ВЕРНУТЬ ПАНЕЛИ ИНСТРУМЕНТОВ В ПЕРВОНАЧАЛЬНОЕ СОСТОЯНИЕ?

Для этого воспользуйтесь диалоговым окном Настройка и сбросьте каждую панель инструментов вручную. Также можно обратиться к следующей процедуре.

```
Sub ResetAllToolbars()
    For Each tb In CommandBars
        If tb.Type = msoBarTypeNormal Then
            If tb.BuiltIn Then tb.Reset
        End If
    Next tb
End Sub
```

Заметим, что эта процедура удаляет все изменения в панелях инструментов, даже те, которые внесены надстройками.

РАССКАЖИТЕ О СПОСОБАХ ОТОБРАЖЕНИЯ СОБСТВЕННОГО МЕНЮ ПРИ АКТИВИЗАЦИИ ОПРЕДЕЛЕННОЙ РАБОЧЕЙ КНИГИ.

Необходимо воспользоваться событиями `WorkbookActivate` и `WorkbookDeactivate`. Другими словами, следует создать процедуру, расположенную в модуле кода объекта `ЭтаКнига`, которая скрывает меню, когда рабочая книга деактивизируется, и отображает меню, когда рабочая книга активизируется.

КАК ДОБАВИТЬ РАЗДЕЛИТЕЛЬ МЕЖДУ ДВУМЯ КНОПКАМИ НА ПАНЕЛИ ИНСТРУМЕНТОВ?

Установите свойство BeginGroup элемента управления, расположенного после разделителя, в значение True.

КАК ОТОБРАЗИТЬ ФЛАЖОК НАПРОТИВ ОПЦИИ МЕНЮ?

Флажок, отображаемый возле опции меню, управляется с помощью свойства State. Следующий оператор отображает флажок возле опции меню My Item.

```
CommandBars(1).Commands("MyMenu"). _
  Commands("My Item").State = msoButtonDown
```

Для того чтобы сбросить флажок возле опции меню, необходимо установить свойство State в значение msoButtonUp.

Я СЛУЧАЙНО УДАЛИЛ НЕКОТОРЫЕ ОПЦИИ МЕНЮ РАБОЧЕГО ЛИСТА И ТЕПЕРЬ НЕ МОГУ ИХ ВЕРНУТЬ. ПЕРЕЗАПУСК EXCEL ПРОБЛЕМУ НЕ РЕШАЕТ

Выберите Сервис⇨Настройка и перейдите в диалоговом окне Настройка на вкладку Панели инструментов. Выберите Строка меню листа и щелкните на кнопке Сброс.

КАК ОТКЛЮЧИТЬ ОТОБРАЖЕНИЕ ВСЕХ КОНТЕКСТНЫХ МЕНЮ?

Для этого воспользуйтесь следующей процедурой.

```
Sub DisableAllShortcutMenus()
    Dim cb As CommandBar
    For Each cb In CommandBars
        If cb.Type = msoBarTypePopup Then _
          cb.Enabled = False
    Next cb
End Sub
```

СУЩЕСТВУЕТ ЛИ СПОСОБ ОТКЛЮЧИТЬ КОНТЕКСТНОЕ МЕНЮ, КОТОРОЕ ОТОБРАЖАЕТСЯ ПРИ ЩЕЛЧКЕ ПРАВОЙ КНОПКОЙ МЫШИ НА ПАНЕЛИ ИНСТРУМЕНТОВ?

Да. Для этого рекомендуется воспользоваться следующим оператором.

```
CommandBars("Toolbar List").Enabled = False
```

Приложение А

Информационные ресурсы, посвященные Excel

Если информация, изложенная в этой книге, оказалась вам полезной, то я выполнил свою задачу. Но книга, к сожалению, не является исчерпывающим источником информации. Кроме того, время от времени возникают новые вопросы, поэтому всегда необходимо оставаться в курсе событий. Таким образом, я систематизировал список ресурсов, которые помогут читателям повысить свою квалификацию как разработчика приложений для Excel. Эти ресурсы разделены на три категории.

- Техническая поддержка со стороны компании Microsoft.
- Группы новостей.
- Web-узлы.

Техническая поддержка со стороны компании Microsoft

Техническая поддержка — это распространенный термин. Он означает поддержку производителя программного обеспечения. В данном случае речь идет о поддержке, непосредственно предоставляемой компанией Microsoft. Техническая поддержка Microsoft реализуется в нескольких формах.

Варианты поддержки

Для того чтобы ознакомиться с доступными вариантами поддержки, выберите команду Справка⇨О программе. После этого щелкните на кнопке Поддержка. В результате будет открыто диалоговое окно, которое содержит список типов технической поддержки, предоставляемых Microsoft. В число этих типов входит платная и бесплатная поддержка.

Опыт многих пользователей говорит о том, что *стандартную поддержку по телефону* необходимо использовать в качестве последнего средства. Высока вероятность того, что звонок в службу поддержки приведет к получению большого счета за телефонные переговоры (если вообще удастся дозвониться). Вам придется долгое время находиться на линии, при этом нет гарантии, что необходимый ответ будет таки получен.

Об URL, перечисленных в этом приложении

Как известно, Internet — динамичная система, которая изменяется очень быстро. Web-узлы часто подвергаются реорганизации (особенно те из них, которые находятся в домене `microsoft.com`). Таким образом, конкретные URL, указанные в этом приложении, могут оказаться недоступными. На момент написания книги каждый URL указывал на реальную страницу, но возможно, что к моменту, когда вы будете ее читать, часть этих адресов окажется недействительной.

На самом деле, те, кто отвечают на подобные телефонные звонки, могут пояснить вам только самые простые вопросы. Однако решить такие вопросы можно и другими путями.

База знаний Microsoft

Наиболее эффективный метод решения проблемы вы найдете в Microsoft Knowledge Base (База знаний Microsoft). Это основной источник информации о продуктах компании Microsoft — обширная база данных, которая поддерживает поиск информации и состоит из десятков тысяч подробных статей, содержащих техническую информацию, а также списков ошибок, исправлений и т.д.

С помощью Internet можно получить бесплатный и неограниченный доступ к ресурсам базы знаний. Эта база данных расположена по адресу `http://support.microsoft.com/`.

Начальная страница Microsoft Excel

Официальная страница Microsoft Excel находится по адресу `http://www.microsoft.com/office/excel/`.

Средства Microsoft Office

Для получения дополнительной информации и поддержки приобретенных продуктов, а также получения надстроек и другой информации об Office (в состав которого входит Excel), обратитесь по следующему адресу: `http://office.microsoft.com/`.

Группы новостей

Служба Usernet — это сеть в пределах Internet, которая предоставляет доступ к тысячам групп новостей, посвященных определенным темам. В этих конференциях можно пообщаться с людьми, у которых общие интересы. Существуют тысячи конференций, посвященных всем темам, которые только можно себе представить (имеются группы новостей, посвященные темам, которые и представить себе нельзя). Обычно на вопросы, размещенные в группах новостей, ответ дается в течение 24 часов (конечно, если вопрос задан таким образом, что у подписчиков возникает желание отвечать).



Кроме соединения с Internet, вам также необходимо иметь специальное программное обеспечение, предназначенное для чтения групп новостей. Почтовый клиент Microsoft Outlook Express является неплохим вариантом такого программного обеспечения.

Группы новостей, посвященные электронным таблицам

Основной группой новостей, посвященной электронным таблицам, является конференция `comp.apps.spreadsheet`. Она предназначена для пользователей всех электронных таблиц, но около 90% сообщений этой конференции посвящены Excel. Хотите бесплатный совет? Не связывайтесь с этой конференцией и переходите непосредственно к конференциям Microsoft.

Конференции Microsoft

Microsoft поддерживает большое количество групп новостей, включая посвященные Excel конференции. Если ваш провайдер не предоставляет доступ к конференциям Microsoft, то обратиться к ним можно непосредственно с помощью браузера, посетив сервер новостей компании Microsoft. Этот сервер расположен по адресу `msnews.microsoft.com`.

В табл. А.1 перечислены ключевые конференции, которые можно найти на этом сервере.

Таблица А.1. Группы новостей на сервере Microsoft, посвященные Excel

Конференция	Тема
`microsoft.public.excel.programming`	Программирование в Excel с помощью VBA или XLM
`microsoft.public.excel.123quattro`	Преобразование листов Lotus 1-2-3 или Quattro Pro в листы Excel
`microsoft.public.excel.worksheet.functions`	Функции рабочих листов
`microsoft.public.excel.charting`	Создание диаграмм с помощью Excel
`microsoft.public.excel.printing`	Печать из Excel
`microsoft.public.excel.queryDAO`	Использование Microsoft Query и Data Access Objects (DAO) в Excel
`microsoft.public.excel.datamap`	Использование средства Excel Data Map
`microsoft.public.excel.crashesGPFs`	Помощь при возникновении ошибки General Protection Fault и аварийном завершении работы системы
`microsoft.public.excel.misc`	Общие темы, оказывающиеся за пределами тематических разделов
`microsoft.public.excel.links`	Использование ссылок в Excel
`microsoft.public.excel.macintosh`	Вопросы, связанные с Excel для Macintosh
`microsoft.public.excel.interopole`	OLE, DDE и другие вопросы взаимодействия приложений
`microsoft.public.excel.setup`	Настройка и установка Excel
`microsoft.public.excel.templates`	Шаблоны Spreadsheet Solutions и другие файлы XLT
`microsoft.public.excel.sdk`	Вопросы, касающиеся среды разработки Excel Software Development Kit

Поиск в группах новостей

Многие пользователи не знают, что в группах новостей можно проводить поиск по ключевым словам. Часто этот путь является отличной альтернативой размещению вопроса в конференции новостей, так как ответ можно получить немедленно. Поиск в группах новостей выполняется по следующему адресу.

```
http://groups.google.com/
```

Ранее поиск в группах новостей выполнялся на узле `Deja.com`. Недавно этот узел был закрыт, а все сведения о группах новостей приобретены компанией Google.

Советы по написанию вопросов в группу новостей

1. Удостоверьтесь, что на этот вопрос еще не отвечали. Проверьте содержимое документов FAQ (если они существуют) и выполните поиск на узле Google (дополнительная информация приводится в разделе "Поиск в группах новостей" в этом приложении).
2. Укажите тему сообщения. Сообщение с темой “Помогите!” и “Вопрос по Excel” привлечет к себе намного меньше внимания, чем сообщение с темой “Код VBA для изменения размеров диаграммы в Excel 2002”.
3. Укажите название и версию программного продукта, используемого вами для работы с электронными таблицами. В большинстве случаев ответ на вопрос будет зависеть от используемой версии Excel.
4. Удостоверьтесь, что вопрос задан достаточно конкретно.
5. Вопрос должен быть краток, желательно, чтобы он находился в рамках основной темы. Кроме того, он должен содержать достаточно информации для получения необходимого ответа.
6. Укажите список действий, которые предпринимались для получения решения.
7. Вопрос должен размещаться в подходящей группе новостей, а рассылка сообщений допустима только в том случае, если сообщение касается нескольких конференций.
8. Не используйте только верхний или только нижний регистр. Проверьте грамматику и орфографию сообщения.
9. Не вкладывайте в сообщение файлы.
10. Не используйте формат HTML.
11. Если необходимо получить ответ по электронной почте, не используйте “антиспам”-адреса, которые требуют модификации адреса перед отправкой. Зачем заставлять выполнять лишнюю работу того, кто и так пытается оказать услугу?

Как производить такой поиск? Предположим, у вас возникла проблема с элементом управления `ListBox`, который расположен в диалоговом окне `UserForm`. В качестве параметров поиска можно задать следующие ключевые слова: `Excel`, `ListBox` и `UserForm`. Поисковая система Google найдет десятки сообщений в группах новостей, которые касаются указанных тем. Просмотр результатов поиска может занять длительное время, но высока вероятность того, что в результате анализа все же будет получен необходимый ответ.

Web-узлы

В Internet расположены тысячи узлов, которые посвящены Excel. Приведем несколько самых посещаемых ресурсов.

Spreadsheet Page

Это собственный Web-узел автора книги. Если отбросить ложную скромность, то данный узел можно назвать лучшим среди тех, которые предоставляют информацию для разработчиков. Он содержит файлы, советы разработчикам, инструкции по доступу к скрытым возможностям Excel, а также обширный список ссылок на другие узлы, посвященные электронным таблицам. Помимо этого, на узле приводится информация о книгах автора и даже шутки об электронных таблицах. Данный узел расположен по адресу `http://j-walk.com/ss/`

Этот узел содержит список ошибок, которые найдены в книгах об Excel, включая книгу, которую вы читаете. (Да, в этой книге тоже имеются ошибки!)

Web-узел Чипа Пирсона, посвященный Excel

Чип является главой Pearson Software Consulting. Его узел содержит ряд полезных примеров использования языка VBA, а также способом применения формул. Этот узел расположен по адресу `http://www.cpearson.com`.

Web-узел Стивена Буле, посвященный Excel

Стивен является разработчиком приложений Excel и проживает в Великобритании. Он является главой Business Modelling Solutions, Ltd. Его узел содержит интересные примеры кода VBA для Excel, включая раздел "Считалось, что это невозможно". Данный узел расположен по адресу `http://www.bmsltd.co.uk/excel/`.

Список часто задаваемых вопросов, касающихся электронных таблиц

Некоторые группы новостей имеют собственные списки часто задаваемых вопросов. Они составляют для того, чтобы предотвратить повторяющиеся вопросы. Список часто задаваемых вопросов конференции `comp.apps.spreadsheet` расположен по адресу `http://www.faqs.org/faqs/spreadsheets/faq`.

Приложение Б

Справочник по функциям и операторам VBA

Это приложение содержит полный список всех операторов VBA и встроенных функций. Дополнительная информация приводится в интерактивном справочном руководстве Excel.

В Excel 2002 и Excel 2003 новые операторы VBA не были добавлены.

Таблица Б.1. Список операторов VBA

Оператор	Действие
`AppActivate`	Активизирует окно приложения
`Beep`	Выдает звуковой сигнал с помощью встроенного в компьютер динамика
`Call`	Передает управление другой процедуре
`ChDir`	Меняет текущую папку
`ChDrive`	Меняет текущий диск
`Close`	Закрывает текстовый файл
`Const`	Объявляет значение константы
`Date`	Устанавливает текущую системную дату
`Declare`	Объявляет ссылку на внешнюю процедуру в библиотеке DLL
`DefBool`	Устанавливает Boolean в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
`DefByte`	Устанавливает Byte в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
`DefDate`	Устанавливает Date в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
`DefDec`	Устанавливает Decimal в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
`DefDouble`	Устанавливает Double в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
`DefInt`	Устанавливает Integer в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
`DefLng`	Устанавливает Long в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
`DefObj`	Устанавливает Object в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов

Оператор	Действие
`DefSng`	Устанавливает Single в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
`DefStr`	Устанавливает String в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
`DefVar`	Устанавливает Variant в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
`DeleteSetting`	Удаляет параметр или раздел приложения из системного реестра
`Dim`	Объявляет массив
`Do-Loop`	Цикл
`End`	Если используется сам по себе, то завершает выполнение программы; кроме того, оператор `End` используется для завершения блоков операторов, начатых с помощью операторов `If`, `Sub`, `Function`, `Property`, `Type` и `Select`
`Enum*`	Объявляет тип перечисления
`Erase`	Повторно инициализирует массив
`Error`	Имитирует определенное ошибочное состояние
`Event*`	Объявляет событие, определенное пользователем
`Exit Do`	Выход из блока операторов `Do-Loop`
`Exit For`	Выход из блока операторов `Do-For`
`Exit Function`	Выход из процедуры `Function`
`Exit Property`	Выход из процедуры свойства
`Exit Sub`	Выход из подпрограммы
`FileCopy`	Копирует файл
`For Each-Next`	Цикл
`For-Next`	Цикл
`Function`	Объявляет имя и аргументы процедуры `Function`
`Get`	Читает данные из текстового файла
`GoSub...Return`	Осуществляет передачу управления
`GoTo`	Осуществляет передачу управления
`If-Then-Else`	Выполняет условный оператор
`Implements*`	Указывает интерфейс или класс, которые будут реализованы в модуле класса
`Input #`	Читает данные из текстового файла с последовательным доступом
`Kill`	Удаляет файл с диска
`Let`	Присваивает значение выражения переменной или свойству
`Line Input #`	Читает строку данных из текстового файла с последовательным доступом
`Load`	Загружает, но не отображает объект
`Lock...Unlock`	Управляет доступом к текстовому файлу
`Lset`	Выравнивает строку по левому краю в пределах строковой переменной
`Mid`	Замещает строку символов другими символами
`MkDir`	Создает новую папку
`Name`	Переименовывает файл или папку

Окончание табл. Б.1

Оператор	Действие
`On Error`	Передает управление в случае возникновения ошибки
`On...GoSub`	Выполняет условную передачу управления
`On...GoTo`	Выполняет условную передачу управления
`Open`	Открывает текстовый файл
`Option Base`	Меняет нижний предел, принятый по умолчанию
`Option Compare`	Объявляет режим сравнения, используемый по умолчанию при сравнении строк
`Option Explicit`	Требует объявления всех переменных в модуле
`Option Private`	Указывает, что весь модуль имеет область действия `Private`
`Print #`	Записывает данные в файл с последовательным доступом
`Private`	Объявляет локальный массив или переменную
`Property Get`	Объявляет имя и аргументы процедуры `Property Get`
`Property Let`	Объявляет имя и аргументы процедуры `Property Let`
`Property Set`	Объявляет имя и аргументы процедуры `Property Set`
`Public`	Объявляет общедоступную переменную или массив
`Put`	Записывает переменную в текстовый файл
`RaiseEvent`	Провоцирует возникновение события, определенного пользователем
`Randomize`	Инициализирует генератор случайных чисел
`ReDim`	Меняет размерность массива
`Rem`	Указывает строку комментария (так же, как и апостроф ['])
`Reset`	Закрывает все открытые текстовые файлы
`Resume`	Продолжает выполнение после завершения процедуры обработки ошибки
`RmDir`	Удаляет пустую папку
`Rset`	Выравнивает строку по правому краю в строковой переменной
`SaveSetting`	Сохраняет или создает параметр приложения в системном реестре
`Seek`	Устанавливает позицию для доступа в текстовом файле
`Select Case`	Условная обработка операторов
`SendKeys`	Отправляет комбинацию клавиш активному окну
`Set`	Назначает ссылку на объект переменной или свойству
`SetAttr`	Изменяет информацию об атрибутах файла
`Static`	Изменяет размерность массива, сохраняя данные
`Stop`	Останавливает выполнение программы
`Sub`	Объявляет имя и аргументы процедуры `Sub`
`Time`	Устанавливает системное время
`Type`	Определяет пользовательский тип
`Unload`	Удаляет объект из памяти
`While...Wend`	Цикл
`Width #`	Устанавливает ширину строки в текстовом файле
`With`	Устанавливает последовательность свойств объекта
`Write #`	Записывает данные в текстовый файл с последовательным доступом

Вызов функций Excel с помощью операторов VBA

Если в VBA не существует функции, эквивалентной функции Excel, то функцию Excel можно вызвать непосредственно в коде VBA. Достаточно перед функцией указать ссылку на объект `WorksheetFunction`. Например, VBA не содержит функции для преобразования радианов в градусы. Так как для этой процедуры в Excel используется отдельная функция, ее можно применять с помощью следующего оператора VBA.

```
Deg = Application.WorksheetFunction.Degrees(3.14)
```

Объект `WorksheetFunction` впервые был предоставлен в Excel 97. Для совместимости с более ранними версиями Excel ссылку на объект `WorksheetFunction` можно опустить, а затем записать оператор следующим образом.

```
Deg = Application.Degrees(3.14)
```

В Excel 2002 и Excel 2003 новые функции VBA не появились.

Таблица Б.2. Список функций VBA

Функция	Действие
`Abs`	Возвращает модуль числа
`Array`	Возвращает массив
`Asc`	Преобразует первый символ строки в значение ASCII
`Atn`	Возвращает арктангенс числа
`CallByName`	Выполняет метод или устанавливает возвращает свойство объекта
`Cbool`	Приводит выражение к типу Boolean
`Cbyte`	Приводит выражение к типу Byte
`Ccur`	Приводит выражение к типу Currency
`Cdate`	Приводит выражение к типу Date
`CDbl`	Приводит выражение к типу Double
`Cdec`	Приводит выражение к типу Decimal
`Choose`	Выбирает и возвращает значение из списка аргументов
`Chr`	Преобразует код символа в символ
`Cint`	Приводит выражение к типу Integer
`CLng`	Приводит выражение к типу Long
`Cos`	Возвращает косинус числа
`CreateObject`	Создает объект OLE Automation
`CSng`	Приводит выражение к типу Single
`CStr`	Приводит выражение к типу String
`CurDir`	Возвращает текущую папку
`Cvar`	Приводит выражение к типу Variant
`CVDate`	Приводит выражение к типу Date (предназначена для обеспечения совместимости; использовать не рекомендуется)

Функция	Действие
CVErr	Возвращает определенное пользователем значение ошибки, которое соответствует номеру ошибки
Date	Возвращает текущую системную дату
DateAdd	Добавляет к дате время
DateDiff	Возвращает интервал времени между двумя датами
DatePart	Возвращает указанную часть даты
DateSerial	Преобразует дату в последовательное число
DateVAlue	Преобразует строку в дату
Day	Возвращает число месяца для определенной даты
DDB	Возвращает амортизацию актива
Dir	Возвращает имя файла или папки, которые соответствуют шаблону
DoEvents	Прекращает выполнение, предоставляя операционной системе возможность обрабатывать другие события
Environ	Возвращает строку с названием рабочей среды
EOF	Возвращает `True`, если достигнут конец текстового файла
Err	Возвращает число, которое соответствует номеру ошибки
Error	Возвращает сообщение об ошибке, которое соответствует номеру ошибки
Exp	Возвращает основу натурального логарифма (e), возведенного в степень
FileAttr	Возвращает режим текстового файла
FileDateTime	Возвращает дату и время последней модификации файла
FileLen	Возвращает размер файла в байтах
Filter	Возвращает отфильтрованное подмножество массива символов
Fix	Возвращает целую часть числа
Format	Отображает сообщение в определенном формате
FormatCurrency*	Возвращает выражение в формате валюты (указывается настройками системы)
FormatDateTime	Отображает выражение в формате даты и времени
FormatNumber	Отображает выражение в формате числа
FormatPercent	Возвращает процентное выражение
FreeFile	Возвращает следующий доступный номер файла для работы с текстовыми файлами
FV	Возвращает будущее значение ежегодной ренты
GetAllSettings	Возвращает список параметров и разделов системного реестра
GetAttr	Возвращает код, представляющий атрибуты файла
GetObject	Получает из файла объект OLE Automation
GetSetting	Возвращает определенный параметр из раздела приложения системного реестра
Hex	Преобразует десятичное число в шестнадцатеричный формат
Hour	Возвращает час времени
IIF	Оценивает выражение и возвращает одну из двух частей

Функция	Действие
Input	Возвращает символ из текстового файла с последовательным доступом
InputBox	Отображает окно, запрашивающее данные у пользователя
InStr	Возвращает позицию строки в другой строке
InStrRev*	Возвращает позицию строки в другой строке, начиная с конца
Int	Возвращает целую часть числа
Ipmt	Возвращает объем процентных выплат за определенный период ежегодной ренты
IRR	Возвращает внутреннюю ставку прибыли для последовательности денежных потоков
IsArray	Возвращает значение True, если переменная является массивом
IsDate	Возвращает значение True, если переменная является датой
IsEmpty	Возвращает значение True, если переменная не инициализирована
IsError	Возвращает значение True, если выражение является значением ошибки
IsMissing	Возвращает значение True, если необязательный аргумент не передавался в процедуру
IsNull	Возвращает значение True, если выражение содержит значение Null
IsNumeric	Возвращает значение True, если выражение рассматривается как число
IsObject	Возвращает значение True, если выражение ссылается на объект OLE Automation
Join	Комбинирует строки, находящиеся в массиве
LBound	Возвращает наименьшее значение размерности массива
LCase	Возвращает строку, преобразованную в нижний регистр
Left	Возвращает указанное количество символов строки, начиная слева
Len	Возвращает длину строки в символах
Loc	Возвращает текущую позицию чтения и записи в текстовом файле
LOF	Возвращает длину открытого текстового файла в байтах
Log	Возвращает натуральный логарифм числа
LTrim	Возвращает копию строки без начальных пробелов
Mid	Возвращает указанное количество символов строки
Minute	Возвращает минуту времени
MIRR	Возвращает модифицированную внутреннюю ставку прибыли для последовательности периодических денежных потоков
Month	Возвращает месяц даты
MonthName	Возвращает строку, содержащую названия месяца
MsgBox	Отображает модальное окно сообщения
Now	Возвращает текущие системные время и дату
NPer	Возвращает количество периодов ежегодной ренты
NPV	Возвращает общее текущее значение инвестиций
Oct	Преобразует десятичное значение в восьмеричное

Функция	**Действие**
Partition	Возвращает строку, которая представляет диапазон, содержащий значение
Pmt	Возвращает значение выплат для ежегодной ренты
Ppmt	Возвращает значение выплат основной суммы ежегодной ренты
PV	Возвращает текущее значение ежегодной ренты
QBcolor	Возвращает код цвета RGB
Rate	Возвращает процентную ставку для периода ежегодной ренты
Replace*	Возвращает строку, в которой подстрока замещается другой строкой
RGB	Возвращает число, представляющее значение цвета RGB
Right	Возвращает определенное количество символов, начиная с правого края строки
Rnd	Возвращает случайное число между 0 и 1
Round	Возвращает округленное число
RTrim	Возвращает копию строки без граничных пробелов
Second	Возвращает секунды указанного времени
Seek	Возвращает текущую позицию в текстовом файле
Sgn	Возвращает целое число, которое обозначает знак числа
Shell	Запускает программу
Sin	Возвращает синус указанного числа
SLN	Возвращает прямое обесценивание актива за период времени
Space	Возвращает строку с указанным количеством пробелов
Spc	Размещает результат при записи в файл
Split*	Возвращает одномерный массив, который содержит количество подстрок
Sqr	Возвращает квадратный корень числа
Str	Возвращает строковое представление числа
StrComp	Возвращает значение, которое указывает результат сравнения строк
StrConv	Возвращает преобразованную строку
String	Возвращает повторяющийся символ или строку
StrReverse*	Возвращает строку с обратным порядком символов
Switch	Оценивает список бинарных выражений и возвращает значение, связанное с первым выражением, которое равно True
SYD	Возвращает амортизацию актива за период времени
Tab	Размещает результат при записи в файл
Tan	Возвращает тангенс числа
Time	Возвращает текущее системное время
Timer	Возвращает количество секунд, которые прошли, начиная с полуночи
TimeSerial	Возвращает время для указанного часа, минуты и секунды
TimeValue	Преобразует строку в последовательное число времени
Trim	Возвращает строку без начальных и/или завершающих пробелов
TypeName	Возвращает строку, которая описывает тип данных переменной
Ubound	Возвращает наибольшую размерность массива

Окончание табл. Б.2

Функция	Действие
UCase	Преобразует строку в верхний регистр
Val	Возвращает число, которое получено из начальных цифр строки
VarType	Возвращает значение, указывающее подтип переменной
Weekday	Возвращает число, указывающее день недели
WeekdayName*	Возвращает строку, содержащую название дня недели
Year	Возвращает год указанной даты

* Не доступно в Excel 97 и более ранних версиях программы.

Приложение В

Коды ошибок VBA

Это приложение содержит коды всех ошибок, которые можно зафиксировать и обработать. Данная информация пригодится также при обработке ошибок. Для получения дополнительных сведений обратитесь к интерактивному справочному руководству Excel.

Код ошибки	Описание
3	`Return` без `GoSub`
5	Неправильный вызов процедуры или аргумент
6	Переполнение (например, значение слишком велико для типа Integer)
7	Недостаточно памяти. Данная ошибка редко вызывается недостаточным объемом физической памяти, установленной в системе. Как правило, ошибка указывает на ограниченный объем памяти, который используется Excel или Windows (например, область памяти, которая применяется для хранения изображений или собственных форматов)
9	Элемент за пределами диапазона. Это сообщение об ошибке выводится тогда, когда именованный элемент не найден в коллекции объектов. Например, если используется код `Sheets("Sheet2")`, а лист `Sheet2` не существует
10	Массив фиксирован или временно заблокирован
11	Деление на ноль
13	Несоответствие типов
14	Недостаточный размер строки
16	Выражение слишком сложное
17	Невозможно выполнить указанную операцию
18	Возникло прерывание со стороны пользователя. Эта ошибка генерируется, если пользователь прерывает выполнение макроса, щелкнув на кнопке Отмена
20	Продолжение работы без ошибки. Это сообщение обычно означает, что перед процедурой обработки ошибки пропущен оператор `Exit Sub`
28	Недостаточный размер стека
35	Подпрограмма или функция не определены
47	Слишком много клиентов библиотеки DLL
48	Ошибка загрузки DLL
49	Неверное соглашение о доступе к DLL
51	Внутренняя ошибка
52	Неверное имя файла или номер
53	Файл не найден
54	Неверный режим файла
55	Файл уже открыт

Продолжение таблицы

Код ошибки	Описание
57	Ошибка ввода/вывода устройства
58	Файл уже существует
59	Неверная длина записи
61	Диск переполнен
62	Достигнут конец файла
63	Неверный номер записи
67	Слишком много файлов
68	Устройство не доступно
70	В доступе отказано
71	Диск не готов
74	Невозможно переименовать на другой диск
75	Ошибка доступа к папке/файлу
76	Папка не найдена
91	Объектная переменная или переменная блока `With` не установлена. Эта ошибка возникает, если не воспользоваться методом `Set` в начале оператора создания объектной переменной. Кроме того, такая ошибка возникает, когда создается ссылка на объект рабочего листа (например, `ActiveCell`), а в это время активен лист диаграммы
92	Цикл `For` не инициализирован
93	Неверная строка шаблона
94	Неверное использование значения `Null`
96	Невозможно принять событие объекта, так как объект уже отправил события максимальному количеству получателей
97	Невозможно вызвать дружественную функцию объекта, который не является экземпляром определяющего класса
98	Свойство или метод не могут содержать ссылку на закрытый объект — ни в виде аргумента, ни в виде возвращаемого значения
321	Неверный формат файла
322	Невозможно создать необходимый временный файл
325	Неверный формат файла ресурса
380	Неверное значение свойства
381	Неверный индекс массива свойств
382	`Set` не поддерживается во время выполнения
383	`Set` не поддерживается (свойство предназначено только для чтения)
385	Необходим индекс массива свойств
387	`Set` не разрешен
393	`Get` не поддерживается во время выполнения
394	`Get` не поддерживается (свойство предназначено только для записи)
422	Свойство не найдено
423	Свойство или метод не найдены
424	Необходим объект. Эта ошибка возникает, если текст после точки не распознается как объект

Код ошибки	Описание
429	Компонент ActiveX не может создавать объекты (зачастую вызвано проблемами регистрации библиотеки, на которую ссылается приложение)
430	Класс не поддерживает автоматизацию или ожидаемый интерфейс
432	Имя файла или имя класса не найдены во время автоматизации
438	Объект не поддерживает это свойство или метод
440	Ошибка средства автоматизации
442	Связь с библиотекой типов или библиотекой объектов для удаленного процесса утеряна. Щелкните на кнопке OK для удаления ссылки
443	Объект Automation не содержит значения, принятого по умолчанию
445	Объект не поддерживает это действие
446	Объект не поддерживает именованные аргументы
447	Объект не поддерживает текущие локальные установки
448	Именованный объект не найден
449	Аргумент обязательный
450	Неверное количество аргументов или неверная установка свойства
451	Процедура `Property Let` не определена и процедура `Property Get` не возвращает объект
452	Неверный порядковый номер
453	Указанная функция DLL не найдена
454	Ресурс кода не найден
455	Ошибка блокировки ресурса кода
457	Этот раздел уже связан с элементом коллекции
458	Переменная имеет тип Automation, не поддерживаемый в Visual Basic
459	Объект или класс не поддерживают набор событий
460	Неверный формат содержимого буфера обмена
461	Метод или член данных не найдены
462	Удаленный сервер не существует или не доступен
463	Класс не зарегистрирован в локальном компьютере
481	Неверное изображение
482	Ошибка принтера
735	Невозможно сохранить файл в папке `TEMP`
744	Искомый текст не найден
746	Заменяющее значение слишком длинное
1004	Ошибка, определенная приложением или объектом. Довольно распространенное универсальное сообщение об ошибке. Данная ошибка возникает тогда, когда ошибка генерируется не в VBA. Другими словами, ошибка определяется в Excel (или в другом объекте) и передается в VBA. Также эта ситуация возникает в случае, если ошибка генерируется специально (для этого используется метод `Raise` объекта `Err`), но она не определена в VBA

Приложение Г

Содержимое компакт-диска

Это приложение описывает содержимое прилагаемого к книге компакт-диска.

Системные требования

Убедитесь, что ваш компьютер удовлетворяет минимальным системным требованиям, представленным в этом разделе. Если ваш компьютер не удовлетворяет нескольким требованиям, то, скорее всего, у вас возникнут проблемы с использованием прилагаемого к книге компакт-диска.

Для Windows 9x, Windows 2000, Windows NT4 (с SP4 или выше), Windows Me или Windows XP:

- Персональный компьютер с процессором с тактовой частотой 120 МГерц или выше.
- По меньшей мере 32 Мбайт оперативной памяти. Рекомендуется 64 Мбайт.
- Устройство чтения компакт-дисков.

Использование компакт-диска в Windows

Для установки элементов с компакт-диска на жесткий диск выполните следующие действия.

1. Вставьте компакт-диск в устройство чтения компакт-дисков.
2. На экране появится диалоговое окно со следующими параметрами.
   - Install: позволяет установить программное обеспечение и/или примеры, созданные автором.
   - Explore: позволяет просмотре структуру компакт-диска.
   - eBook: позволяет ознакомиться с электронной версией английского варианта этой книги.
   - Exit: закрывает окно автозапуска.

Примеры из глав

Каждая из тех глав этой книги, в которой рассматриваются примеры рабочих книг, представлена на компакт-диске отдельной папкой. Например, файлы примеров из главы 3 расположены в папке `Examples\chap 03\(Примеры\Глава 03)`.

Ниже приводится список примеров рабочих книг каждой главы. Все примеры сопровождаются кратким описанием.

Глава 3

`array examples.xls` — примеры формул массивов.

count and sum.xls — примеры формул суммирования и подсчета.

megaformula-1.xls — пример формулы удаления среднего имени (обычные формулы).

megaformula-2.xls — пример формулы для удаления среднего имени (мегаформула).

megaformula-3.xls — пример формулы удаления среднего имени с помощью пользовательской функции VBA.

named formula.xls — примеры использования именованных формул.

xdate.exe — инструкции по установке надстройки Extended Date Functions.

Глава 4

\amortizations — это подпапка, в которой расположены файлы примера приложения, рассчитывающего параметры займа.

employee list.xml — простой XML-файл, содержащий сведения о сотрудниках фирмы.

message.xml — XML-файл, содержащий данные автоматически генерируемого почтового сообщения.

Глава 6

controls on sheet.xls — пример добавления на рабочий лист элементов управления.

Глава 7

comment objects.xls — примеры кода VBA, предназначенного для управления объектами Comment.

hello world.xls — простой пример VBA-кода, обращающегося к пользователю с приветствием.

Глава 8

timing test.xls — процедура, демонстрирующая разницу в скорости обработки разных типов данных.

Глава 9

sheet sorter.xls — приложение для сортировки рабочего листа.

Глава 10

commission.xls — содержит различные версии функции Commission, предназначенной для подсчета размера комиссионных с продажи.

draw.xls — содержит функцию Draw, которая случайным образом выбирает ячейку из диапазона.

key press.xls — демонстрирует использование функций API для определения состояния клавиш <Shift>, <Ctrl> и <Alt>.

month names.xls — демонстрирует функцию MonthNames, которая возвращает массив.

mysum.xls — демонстрирует функцию MySum, которая эмулирует функцию Excel СУММ.

`reverse.xls` — демонстрирует функцию `Reverse`, которая возвращает значение ошибки, если ее аргумент не является строкой.

`uppercase.txt` — содержит функцию `UpCase`, которая эмулирует функцию Excel `ПРОПИСН`.

`win32api.xls` — содержит объявление функций и констант Windows API.

`windows directory.xls` — демонстрирует функцию `ShowWindowsDir`, которая использует функцию API для отображения имени папки Windows.

Глава 11

`about selection.xls` — содержит процедуру, которая описывает текущий выделенный диапазон.

`auto average.xls` — демонстрирует добавление функции `AVERAGE`, подобной по возможностям команде Автосумма. Когда эта рабочая книга открыта, предоставляется доступ к новой опции меню: Сервис⇨Enter Average Formula.

`batch processing.xls` — демонстрирует пакетную обработку файлов. В примере используется три дополнительных файла: `text01.txt`, `text02.txt` и `text03.txt`.

`cell type.xls` — использует функцию `CellType` для определения типа данных ячейки.

`data type.xls` — использует функцию `CellType` для определения типа данных, находящихся в ячейке.

`date and type.xls` — содержит процедуру, отображающую текущие дату и время.

`delete empty rows.xls` — содержит процедуру удаления пустых строк из рабочего листа.

`disk info.xls` — демонстрирует различные функции API, предоставляющие информацию о диске.

`extract element.xls` — демонстрирует использование функции `ExtractElement`.

`file association.xls` — демонстрирует использование функции API, которая возвращает полный путь к приложению, связанному с указанным типом файлов.

`fill range.xls` — содержит две процедуры, которые демонстрируют заполнение диапазона данными из массива.

`find by format.xls` — содержит процедуру выбора ячейки на основе ее форматирования.

`in range.xls` — содержит функцию, которая возвращает значение `True`, если диапазон находится в пределах другого диапазона.

`inputBox.xls` — содержит примеры использования функции VBA `InputBox` для получения информации от пользователя.

`last saved and printed.xls` — содержит функции `LastSaved` и `LastPrinted`, которые получают доступ к встроенным свойствам документа рабочей книги.

`list fonts.xls` — создает список всех установленных шрифтов.

`max all sheets.xls` — содержит функцию `MaxAllSheets`, которая возвращает максимальное значение на всех листах рабочей книги.

`next empty row.xls` — демонстрирует вставку данных в следующую пустую строку рабочей книги.

page count.xls — содержит процедуру, подсчитывающую количество страниц, которые будут напечатаны.

play sound.xls — содержит функцию Alarm, которая проигрывает звуковой файл sound.wav при определенном условии.

printer info.xls — демонстрирует функцию API, которая возвращает информацию об используемом по умолчанию принтере.

random functions.xls — содержит функцию RandomIntegers (которая возвращает массив неповторяющихся случайных целых чисел) и функцию RangeRandomize (которая заполняет диапазон случайными значениями).

range selections.xls — демонстрирует несколько распространенных способов выделения диапазона относительно активной ячейки. После открытия этого файла можно использовать меню Selection Demo.

registry.xls — демонстрирует функции API, которые позволяют считывать и записывать информацию в системном реестре.

select max.xls — содержит процедуру, которая выделяет ячейку с максимальным значением.

selective color.xls — содержит процедуру заливки ячеек на основе их содержимого.

sheet offset.xls — демонстрирует две версии функции SheetOffset.

sorting demo.xls — демонстрирует три процедуры VBA, используемые для сортировки массивов.

sound.xls — демонстрирует функции API, предназначенные для проигрывания звуковых файлов.

sound.wav — звуковой файл, используемый в файле sound.xls.

stat functions.xls — демонстрирует функцию StatFunction.

synchronize sheets.xls — содержит процедуру синхронизации рабочих листов.

toggles.xls — содержит процедуру переключения параметров.

utility functions.xls — содержит следующие функции: FileExists, FileNameOnly, PathExists, RangeNameExists, SheetExists и WorkBookIsOpen.

variant transfer.xls — демонстрирует передачу диапазона ячеек в массив переменного типа, а также передачу массива переменного типа в диапазон ячеек.

video mode.xls — демонстрирует использование функции API, которая возвращает текущее разрешение экрана.

worksheet functions.xls — содержит следующие функции рабочего листа: SheetName, WorkbookName, AppName, CountBetween, LastInColumn, LastInRow и IsLike.

write and read range.xls — содержит процедуры записи данных в диапазон и чтения данных из диапазона.

Глава 12

erase a range.xls — демонстрирует использование функции InputBox для указания очищаемого диапазона.

get a filename.xls — демонстрирует использование метода GetOpenFilename.

get directory.xls — демонстрирует использование функции API для отображения диалогового окна, позволяющего получить от пользователя имя папки.

get user name.xls — содержит пример использования функции InputBox для получения имени пользователя.

Глава 13

controls on sheet.xls — пример использования элементов управления (с панели инструментов Элементы управления) на рабочем листе.

controls on sheet2.xls — еще один пример, демонстрирующий использование элементов управления на рабочем листе.

get name and sex.xls — конечный результат примера, рассмотренного в главе 13.

newcontrols.pag — содержит модифицированные элементы управления, размещенные в окне Toolbox (Панель элементов) редактора VBE. Для импорта этого файла в качестве новой страницы окна щелкните в окне Toolbox правой кнопкой мыши и выберите Import Page (Импорт страницы).

spinbutton events.xls — демонстрирует применение событий, связанных с элементом управления SpinButton.

spinbutton textbox.xls — описывает способы совместного использования элементов управления SpinButton и TextBox.

userform events.xls — демонстрирует применение событий, связанных с диалоговым окном UserForm.

Глава 14

change size.xls — создает диалоговое окно UserForm переменного размера.

fill listbox.xls — демонстрирует два способа добавления опций в список элемента управления ListBox.

listbox activate sheet.xls — демонстрирует, как можно отобразить список имен рабочих листов с помощью элемента управления ListBox

listbox item transfer.xls — предоставляет возможность обмена опциями списка между двумя элементами управления ListBox.

listbox move items.xls — предоставляет пользователю возможность перемещать опции вверх и вниз по списку элемента управления ListBox.

listbox select rows.xls — демонстрирует использование элемента управления ListBox, который содержит несколько столбцов и позволяет выделять строки рабочего листа.

multicolumn listbox.xls — создает элемент управления ListBox, который состоит из нескольких столбцов на основе данных, содержащихся на рабочем листе.

multicolumn listbox2.xls — создает элемент управления ListBox, который состоит из нескольких столбцов на основе данных, хранящихся в массиве.

multiple lists.xls — отображает несколько списков в элементе управления ListBox.

queryclose.xls — демонстрирует методику, предотвращающую закрытие пользователем диалогового окна UserForm с помощью кнопки Закрыть.

refedit.xls — демонстрирует использование элемента управления RefEdit.

selected items.xls — демонстрирует определение выделенных опций элемента управления ListBox.

splash.xls — отображает экран-заставку в момент загрузки рабочей книги.

unique1.xls — заполняет элемент управления ListBox уникальными опциями.

unique2.xls — заполняет элемент управления ListBox уникальными опциями.

userform menus.xls — содержит две простые системы меню, представленные элементами управления CommandButton и ListBox.

zoom and scroll sheets.xls — использует элементы управления диалогового окна для масштабирования и прокрутки рабочего листа.

zoom.xls — использует свойство Zoom для масштабирования диалогового окна.

Глава 15

\dataform — папка надстройки, которая служит заменой диалоговому окну Excel Форма данных. Данная надстройка защищена паролем.

chart in userform.xls — отображает диаграмму в диалоговом окне UserForm.

color picker.xls — содержит функцию, которая предоставляет возможность выбирать цвет из диалогового окна.

modeless userform.xls — демонстрирует создание немодальных диалоговых окон UserForm, которые отображают информацию об активной ячейке.

modeless userform2.xls — демонстрирует создание более сложного немодального диалогового окна UserForm, которое отображает информацию об активной ячейке.

multiple buttons.xls — использует единственную процедуру обработки событий в нескольких элементах управления.

my msgbox.xls — содержит функцию MyMsgBox, которая эмулирует функцию VBA MsgBox.

owc chart - simple.xls — создает диаграмму в диалоговом окне UserForm. Для этого используется средство Office Web Components.

owc chart - advanced.xls — создает диаграмму в диалоговом окне UserForm с помощью Office Web Components.

owc spreadsheet.xls — демонстрирует применение элемента управления Spreadsheet, который входит в состав Office Web Components.

progress-1.xls — отображает индикатор выполнения макроса. (Индикатор выполнения макроса инициализируется за пределами диалогового окна UserForm.)

progress-2.xls — демонстрирует другой способ отображения индикатора выполнения макроса. (Индикатор выполнения макроса инициализируется в диалоговом окне UserForm.)

progress-3.xls — демонстрирует другой способ отображения индикатора выполнения макроса. (Индикатор выполнения макроса инициализируется в диалоговом окне UserForm.)

wizard.xls — создает “мастер”, состоящий из нескольких этапов.

Глава 16

\helpsource — содержит исходные HTML-файлы для надстройки Text Tools.

text tools.xls — утилита управления текстом, которая рассматривалась в главе 15 (версия утилиты в виде рабочей книги).

text tools.chm — компилированный файл справочной системы.

undo.xls — демонстрирует один из способов отмены действий, выполненных макросом VBA.

Глава 17

`budget.xls` — создает сводную таблицу на основе базы данных рабочего листа (с помощью кода VBA).

`budget.mdb` — файл Access, используемый при создании свойдной таблицы.

`external db.xls` — создает сводную таблицу на основе внешней базы данных (с помощью кода VBA). Эта рабочая книга использует файл базы данных `budget.mdb`.

`modify pivot.xls` — создает код VBA, предназначенный для внесения изменений в сводную таблицу.

`simple db.xls` — простая база данных рабочего листа, которая используется для демонстрации методов создания сводных таблиц. Эта рабочая книга содержит записанный макрос, а также "подкорректированную" версию макроса.

`survey data.xls` — демонстрирует операцию создания нескольких сводных таблиц для анализа данных опроса.

Глава 18

`animated chart.xls` — пример анимированной диаграммы.

`autofilter chart.xls` — демонстрирует использование средства Автофильтр в диаграмме.

`chart active cell.xls` — вносит изменения в последовательность данных диаграммы на основе активной ячейки.

`chart formatting.xls` — демонстрирует, как можно форматировать диаграмму с помощью макроса.

`chart image map.xls` — создает диаграмму, которая служит в качестве изображения-карты.

`chart in userform.xls` — создает диаграмму "на лету" и отображает ее в пользовательском диалоговом окне.

`climate date.xls` — интерактивная диаграмма, в которой не используются макросы.

`clock chart.xls` — содержит диаграмму, которая имитирует аналоговые (стрелочные) часы.

`combobox chart.xls` — демонстрирует использование элемента управления `ComboBox` для внесения изменений в последовательности данных диаграммы.

`convert chart to picture.xls` — демонстрирует, как диаграмму можно преобразовать в изображение.

`create chart.xls` — содержит макрос, записанный при создании диаграммы. Кроме того, в рабочей книге содержится "подкорректированная" версия макроса создания диаграммы.

`dailysales.xls` — демонстрирует саморасширяющуюся диаграмму.

`data labels.xls` — демонстрирует использование диапазона для подписи данных диаграммы.

`events - chart sheet.xls` — управляет событиями диаграммы на листе диаграммы.

`events - embedded.xls` — управляет событиями встроенной на рабочий лист диаграммы.

get chart range.xls — содержит пользовательские функции, которые возвращают объект Range, представляющий данные диаграммы.

hypocycloid – animated.xls — содержит диаграмму, которая отображает гипоциклоидные кривые, разноцветные и анимированные.

hypocycloid.xls — содержит диаграмму, которая отображает гипоциклоидные кривые.

mouseover event – chart sheet.xls — демонстрирует отображение подсказки для объектов диаграммы при наведении на них указателя (для диаграммы на отдельном листе).

mouseover event – embedded.xls — демонстрирует отображение подсказки для объектов диаграммы при наведении на них указателя (для встроенной диаграммы).

multipe charts.xls — отображает несколько диаграмм на листе диаграммы.

printed embedded chart.xls — печатает встроенную диаграмму на отдельном листе бумаги.

size and align charts.xls — демонстрирует выравнивание и изменение размера группы диаграмм.

Глава 19

application events.xls — управляет событиями объекта Application. В этом случае используется немодальное диалоговое окно UserForm, поэтому для загрузки рабочей книги требуется Excel 2000 или более поздняя версия программы.

log workbook open.xls — отслеживает все открытые рабочие книги и записывает сведения о них в текстовый файл.

make formulas bold.xls — выделяет каждую формулу полужирным начертанием.

onkey demo.xls — демонстрирует использование метода OnKey для переопределения клавиш.

selection change.xls — использует событие SelectionChage объекта Worksheet для выделения активной строки и столбца цветом.

track changes in comments.xls — содержит процедуру, которая использует примечания для фиксирования изменений, внесенных в ячейку.

validate entry1.xls — выполняет проверку правильности введенных в ячейки значений. Для этого используется свойство EnableEvents.

validate entry2.xls — проверяет правильность введения значения в ячейку. Свойство EnableEvents не используется.

Глава 20

automate Excel.doc — файл Microsoft Word, который содержит процедуру автоматизации с Excel. Для этого используется файл рабочей книги projectsion.xls.

budget.mdb — база данных Access, используемая в примерах этой главы.

control panel dialogs.xls — отображает любое из 50 системных диалоговых окон.

make memos.xls — демонстрирует применение средства автоматизации в Microsoft Word, которое позволяет создавать заметки на основе данных, хранимых в рабочем листе.

`personalized email — OE sendkey.xls` — демонстрирует отправку почтового сообщения из Excel с помощью Outlook.

`personalized email — Outlook.xls` — демонстрирует отправку почтового сообщения из Excel с помощью Outlook.

`simple ado example.xls` — получает данные из файла Microsoft Access (используется файл `budget.mdb`).

`start calculator.xls` — запускает (или активизирует) приложение Microsoft Calculator (Калькулятор).

`start charmap.xls` — демонстрируется два способа запуска приложения Microsoft Character Map (Таблица символов).

Глава 21

`is addin installed.xla` — содержит код, определяющий правильность установки надстройки.

`text tools.xls` — утилита управления текстом, которая рассматривалась в главе 16.

Глава 22

`autosense.xls` — содержит процедуру создания "чувствительных" панелей инструментов, которые отображаются только тогда, когда активная ячейка находится в определенном диапазоне.

`button styles.xls` — демонстрирует различные способы отображения элемента управления `CommandBarButton`.

`hide and restore.xls` — содержит процедуру скрытия и восстановления панелей инструментов.

`list all controls.xls` — содержит процедуру, отображающую значения свойства `Caption` для всех элементов управления на всех панелях инструментов.

`list commandbars.xls` — содержит процедуру, отображающую имя, индексный номер и тип каждого элемента управления `CommandBar`.

`month list.xls` — демонстрирует использование раскрывающегося списка в элементе управления `CommandBar`.

`number format toolbar.xls` — создает кнопку на панели инструментов, которая отображает строку числового формата активной ячейки.

`show face ids.xls` — упрощает определение значения свойства `FaceId` для изображений.

Глава 23

`add new menu.xls` — содержит процедуру добавления нового меню и опций меню.

`add to Tools menu.xls` — содержит процедуру добавления новых опций в меню **Сервис** в строке меню листа.

`hide menu.xls` — демонстрирует отображение меню только при активности определенной рабочей книги.

`list menu info.xls` — содержит процедуру, отображающую название каждой опции (меню, опции меню и опции подменю) в строке меню листа.

`list shortcut menus.xls` — содержит процедуру, перечисляющую все контекстные меню.

menu maker.xls — демонстрирует простой способ создания меню с помощью информации, содержащейся на рабочем листе.

new menubar.xls — демонстрирует замещение строки меню Excel.

new shortcut menu.xls — содержит процедуру создания нового контекстного меню.

shortcut key.xls — содержит процедуру добавления новой опции меню с указанием комбинации клавиш.

toggle gridlines.xls — отображает меню “переключения” с флажком.

Глава 24

\assistant\formletter.xls — отображает справочную информацию с помощью помощника по Office.

\comments\formletter.xls — отображает справочную информацию с помощью комментариев к ячейкам.

\function\myfunc.xls — отображает описание пользовательских функций.

\htmlhelp\formletter.xls — создает простую справочную систему с помощью службы HTML Help (содержит исходные файлы).

\other\myapp.xls — демонстрирует другие способы отображения справочной информации: с помощью метода Help, окна сообщения и текстового поля.

\textbox\formletter.xls — отображает справочную информацию с помощью элемента управления TextBox, расположенного на рабочем листе.

\userform1\formletter.xls — отображает справочную информацию с помощью элемента управления Label, расположенного в диалоговом окне UserForm.

\userform2\formletter.xls — отображает справочную информацию с помощью “прокручиваемого” элемента управления Label в диалоговом окне UserForm.

\userform3\formletter.xls — отображает справочную информацию с помощью элемента управления ComboBox и элемента управления Label, расположенных в диалоговом окне UserForm.

\worksheet\formletter.xls — демонстрирует методы отображения справочной информации после активизации рабочего листа.

В некоторых примерах главы 24 используется несколько файлов. В большинстве рассмотренных примеров применяются файлы с одинаковыми именами. По этой причине данные каждого примера расположены в отдельной папке.

Глава 25

loan amortization wizard.xla — надстройка-мастер, которая создает график погашения займа с фиксированной процентной ставкой. Эта надстройка не защищена. После установки надстройки доступ к мастеру осуществляется с помощью меню Сервис.

Глава 26

multilingual wizard.xls — простой мастер, позволяющий пользователю выбрать один из трех языков.

Глава 27

does file exist.xls — содержит код проверки существования файла. В коде используются три различных методики.

does path exist.xls — содержит код проверки существования пути. В коде используются две различных методики.

`export import.xls` — содержит процедуры экспорта диапазона в файл CSV, а также импорта данных из файла CSV, начиная с активной ячейки.

`export to HTML.xls` — содержит код экспорта диапазона ячеек в формат HTML.

`export to XML.xls` — содержит код экспорта диапазона ячеек в формат XML.

`filter text file.xls` — содержит процедуру чтения текстового файла (`infile.txt`), которая копирует в файл `output.txt` только строки, соответствующие определенному критерию.

`import more than 256.xls` — содержит процедуру чтения текстового файла и сохранения полученных данных на рабочем листе `Sheet1`. Если строка содержит более 256 столбцов, то дополнительные столбцы сохраняются на дополнительных рабочих листах. Этот пример использует файл `longfile.txt`, который содержит 600 столбцов.

`list files1.xls` — содержит процедуру, отображающую список файлов, которые расположены в определенной папке. Кроме того, отображается размер и дата модификации каждого файла.

`list files2.xls` — содержит процедуру, отображающую список файлов, которые расположены в определенной папке. Помимо этого, отображается размер и дата модификации каждого файла. Данный пример использует объект `FileSearch`.

`longfile.txt` — текстовый файл, используемый в примере `import more than 256.xls`.

`show drive info.xls` — содержит процедуру, использующую объект `FileSystemObject` для получения и отображения различной информации о жестких дисках.

Глава 28

`add 100 buttons.xls` — содержит процедуру, добавляющую 100 элементов управления `CommanButton` в диалоговое окно `UserForm` и создающую для каждого элемента управления свою процедуру обработки события.

`add button and code.xls` — содержит процедуру добавления элемента управления `CommandButton` и процедуры VBA.

`add controls.xls` — демонстрирует добавление элементов управления в диалоговое окно `UserForm` на этапе выполнения.

`add userform.xls` — содержит процедуру создания диалогового окна `UserForm` “на лету”.

`options form.xls` — содержит функцию создания диалогового окна `UserForm` “на лету” (с элемента управления `OptionButton`), после чего функция возвращает целое число, которое указывает на выбор, сделанный пользователем.

`replace module.xls` — содержит процедуру замещения модуля другим модулем. Этот пример использует файл `UserBook.xls`.

`show components.xls` — содержит процедуру, отображающую информацию о каждом компоненте VBA, который входит в активную рабочую книгу.

Глава 29

`csvclass.xls` — содержит модуль класса, который упрощает импорт и экспорт данных с помощью файлов в формате CSV.

`keyboard.xls` — содержит модуль класса, который определяет классы `NumLock` и `CapsLock`.

Предметный указатель

П

Р

С

Т

У

Ф

Х

Ц

Ч

Ш

Э

Я

Научно-популярное издание

Джон Уокенбах

Профессиональное программирование на VBA в Excel 2003

Литературный редактор *О.В. Ожигова*
Верстка *А.В. Плаксюк*
Художественный редактор *С.А. Чернокозинский*
Корректоры *Л.А. Гордиенко*, *Л.В. Пустовойтова*

Издательский дом "Вильямс"
101509, г. Москва, ул. Лесная, д. 43, стр. 1
Изд. лиц. ЛР № 090230 от 23.06.99
Госкомитета РФ по печати

Подписано в печать 25.02.2005. Формат 70x100/16.
Гарнитура Times. Печать офсетная
Усл. печ. л. 64,5. Уч.-изд. л. 45,65
Тираж 3000 экз. Заказ №

Отпечатано с диапозитивов в ФГУП "Печатный двор"
Министерства РФ по делам печати,
телерадиовещания и средств массовых коммуникаций
197110, С.-Петербург, Чкаловский пр., 15